Advanced Workbook

21J19

Contents

1 Take a Quick Test Drive

This lab will take approximately *25 minutes* to complete.

You must complete this lab to be able to complete all of the remaining labs in this course. Do not skip this lab.

The instructor will provide you with a URL to connect to the training account, as well as a user name and password. The remainder of the labs in this workbook assume that you can log in as needed.

1.1 Explore the User Interface

The purpose of this task is simply to make sure you are comfortable navigating around the user interface, and can perform basic tasks like rename a worksheet, turn on code highlighting, and switch your workksheet context.

1.1.1 Make sure you are in the Worksheets section of the interface (this is where you will be by default when you log in).

1.1.2 Double-click the tab at the top of your worksheet, and rename it “Lab 1.”

1.1.3 Review the list of databases in the object browser (on the left-hand side of the worksheet).

1.1.4 Change to the PUBLIC role:

```
USE ROLE PUBLIC;
```

Notice that the list in the object browser changes. The role you are currently using will determine what objects you can access.

1.1.5 Change back to the TRAINING_ROLE role:

```
USE ROLE TRAINING_ROLE;
```


1.1.6 Turn on Code Highlighting. To do this, look for the three dots just to the right of the worksheet context (in the upper right-hand corner of the worksheet). Click on the three dots and select “Turn on Code Highlight.”

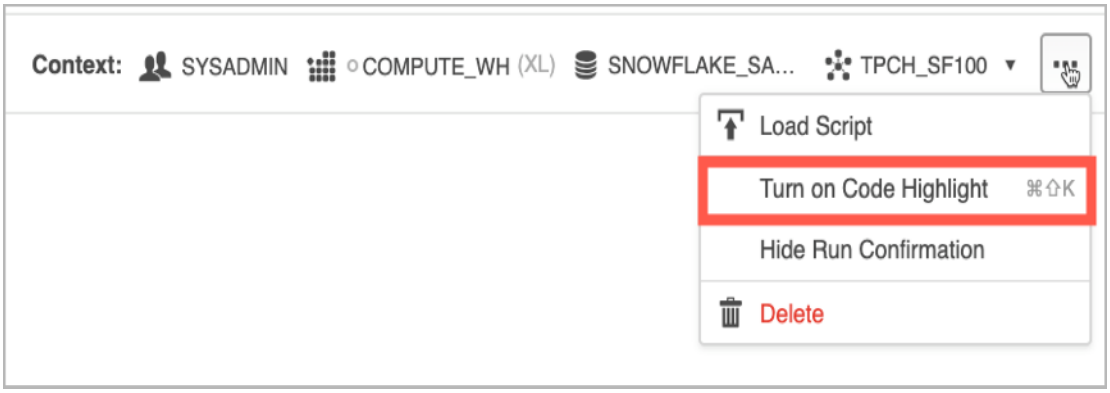

Figure 1: Enable Code Highlight

Now when your cursor is on a line of code, the entire statement will be highlighted. This shows you what will be executed if you run the statement.

1.1.7 Enter the following SQL statement and then click “Run” at the top of the worksheet.

```
SHOW PARAMETERS FOR SESSION;
```

1.1.8 Run the same command again, using the keyboard shortcut. This will be CTRL+RETURN on Windows machines, or CMD+RETURN on Macs.

```
SHOW PARAMETERS FOR SESSION;
```

1.1.9 Quickly explore what is available to you in the top ribbon. When you are done, return to the Worksheets area.

1.2 Create Objects for Course Labs

1.2.1 Run the following commands to create some basic objects that will be used during this course:

```
CREATE DATABASE [login]_db;
CREATE SCHEMA [login]_db.myschema;
CREATE WAREHOUSE [login]_wh INITIALLY_SUSPENDED=TRUE AUTO_SUSPEND=300;
CREATE WAREHOUSE [login]_wh_small WAREHOUSE_SIZE=SMALL INITIALLY_SUSPENDED=TRUE
    AUTO_SUSPEND=300;
CREATE WAREHOUSE [login]_wh_large WAREHOUSE_SIZE=LARGE INITIALLY_SUSPENDED=TRUE
    AUTO_SUSPEND=300;
```


When you create virtual warehouses through the command line, they will be set to automatically resume when needed, unless you set AUTO_RESUME=FALSE.

1.2.2 Run the following commands to set your context (the database, schema, role, and virtual warehouse that will be used in this worksheet). Some of these objects may already be set in your context, but these are the four default elements you can set for every worksheet context.

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_wh_small;
USE DATABASE [login]_db;
USE SCHEMA myschema;
```

1.2.3 Set your default context, so that every time you open a new worksheet these values will automatically be set for you:

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

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

1.2.5 Open a new worksheet and verify that your default context is automatically set when you open a new worksheet. The context for any existing worksheets will not be changed.

1.3 Run Queries on Sample Data

1.3.1 In the left-side object browser, navigate to **SNOWFLAKE_SAMPLE_DATA**, then **TPCH_SF1**.

1.3.2 Right-click the schema name (TPCH_SF1) and select **Set as Context**. This is another way that you can change your worksheet context.

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

1.3.4 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.

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

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

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

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

```
SHOW TABLES;

SELECT COUNT(*) FROM orders;

SELECT * FROM supplier LIMIT 10;

SELECT MAX(o_totalprice) FROM orders;

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

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

1.3.8 View the Query Profile by clicking “Query ID” just above the results pane. Follow the link to get to the Query Profile. Review the information in the Details tab, then select the Profile tab to show more information about the query.

We will view the query profile many times throughout this course, so make sure you know how to get there.

1.3.9 Click the Worksheets icon in the top ribbon to return to the Worksheets area.

2 Time Travel and Cloning

This lab will take approximately *15 minutes* to complete.

2.1 Drop and Undrop Objects

2.1.1 Use the following commands to set up the objects for this lab.

```
USE ROLE TRAINING_ROLE;
CREATE DATABASE IF NOT EXISTS [login]_CLONE_DB;
CREATE OR REPLACE SCHEMA [login]_lab;
USE SCHEMA [login]_lab;
CREATE TABLE nation CLONE training_db.traininglab.nation;
```


2.1.2 Use the `UNDROP` command on a table.

```
SHOW TABLES IN SCHEMA [login]_lab;

DROP TABLE [login]_lab.NATION;

SHOW TABLES IN SCHEMA [login]_lab;

UNDROP TABLE NATION;

SHOW TABLES IN SCHEMA [login]_lab;
```

2.1.3 Use the `UNDROP` command on a schema.

```
SHOW SCHEMAS IN database [login]_CLONE_DB;

DROP SCHEMA [login]_lab;
SHOW SCHEMAS IN database [login]_CLONE_DB;

UNDROP SCHEMA [login]_lab;
SHOW SCHEMAS IN database [login]_CLONE_DB;
```

2.1.4 Use the `UNDROP` command on a database.

```
SHOW DATABASES STARTS WITH '[login]';

DROP database [login]_CLONE_DB;
SHOW DATABASES STARTS WITH '[login]';

UNDROP database [login]_CLONE_DB;
SHOW DATABASES STARTS WITH '[login]';
```

2.2 Explore Time Travel

In an earlier lab, you used Time Travel with the `CLONE` command. In this task, you will explore its use in a query statement.

2.2.1 Set your context.

```
USE DATABASE [login]_CLONE_DB;
USE SCHEMA [login]_lab;
USE WAREHOUSE [login]_wh;
ALTER WAREHOUSE [login]_wh set warehouse_size=small;
```


2.2.2 Make a change to a table, and save the query ID.

```
SELECT N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT
  FROM NATION;

UPDATE NATION
  SET N_NAME ='ERROR'
  WHERE N_NATIONKEY=1;

SET un = last_query_id();
```

2.2.3 Select from the table before and after you made the change.

```
SELECT
  (SELECT N_NAME FROM NATION BEFORE(STATEMENT => $un)
  WHERE N_NATIONKEY=1) ORIGINAL,
  (SELECT N_NAME FROM NATION WHERE N_NATIONKEY=1) "CURRENT";
```

2.2.4 Recover the table to its previous state using a clone.

Often you use the `CLONE` command as part of agile development. It also has a role in data recovery or undoing mistakes.

Restore the `NATION` table to the state it was in prior to the update you made in the previous exercise:

```
SHOW TABLES STARTS WITH 'NATION';

CREATE TABLE NATION_RESTORED CLONE NATION BEFORE(STATEMENT => $un);

SELECT * FROM nation_restored;

ALTER TABLE NATION SWAP WITH NATION_RESTORED;

SELECT (SELECT N_NAME FROM nation_restored
                WHERE  N_NATIONKEY=1) ERROR,
      (SELECT N_NAME FROM nation
                WHERE  N_NATIONKEY=1) RESTORED;
```

3 Querying Between Databases and Accounts

This lab will take approximately *15 minutes* to complete.

3.1 Prepare for the Lab

The purpose of this lab is to show that query performance is not degraded by including shares, clones, or multiple databases.

3.1.1 Open a new worksheet and make sure your default context is set.

3.1.2 Change your warehouse size to medium, and use the schema you created in lab 1:

```
ALTER WAREHOUSE [login]_wh SET WAREHOUSE_SIZE=Medium;
USE WAREHOUSE [login]_wh;
USE SCHEMA [login]_db.myschema;
```

3.1.3 Create two temporary tables using CTAS.

```
CREATE TEMPORARY TABLE customer_ctas AS
   SELECT * FROM snowflake_sample_data.tpch_sf100.customer;

CREATE TEMPORARY TABLE orders_ctas AS
   SELECT * FROM snowflake_sample_data.tpch_SF100.orders;
```

3.1.4 Create two temporary tables using cloning. Note that the cloning is immediate, while the CTAS took time to complete.

```
CREATE TEMPORARY TABLE customer_clone
   CLONE customer_ctas;

CREATE TEMPORARY TABLE orders_clone
   CLONE orders_ctas;
```

3.1.5 Turn off the Query Result Cache, to make sure all statements are executed rather than being taken from cache. Then suspend your warehouse to clear anything in the data cache.

```
ALTER SESSION SET USE_CACHED_RESULT=FALSE;

ALTER WAREHOUSE [login]_wh SUSPEND;
```

NOTE that you may get an error saying that the warehouse cannot be suspended - this means that the warehouse is already suspended. Ignore this message if it occurs in this or subsequent labs.

3.2 JOIN Local Tables

3.2.1 Run the following query to join the two tables you created with CTAS:

```
SELECT c.c_custkey, c.c_name, o.o_orderstatus, o.o_orderpriority
  FROM customer_ctas c JOIN orders_ctas o ON c.c_custkey=o.o_custkey;
```


3.2.2 Run the following query to join the two tables you created by cloning:

```
SELECT c.c_custkey, c.c_name, o.o_orderstatus, o.o_orderpriority
  FROM customer_clone c JOIN orders_clone o ON c.c_custkey=o.o_custkey;
```

3.2.3 Review the query profiles and compare the total execution time and number of rows returned.

The join of the cloned tables may have run faster than the join with the tables created with CTAS. Why might this be?

Look at the “Percentage scanned from cache” in the query profiles. A cloned tables shares micro-partitions with the source table. Because of this, the second JOIN was able to use data that was stored in the data cache on the virtual warehouse.

3.2.4 Suspend your warehouse to clear the data cache.

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

3.2.5 Re-run the JOIN of the cloned tables, and verify execution time and number of rows returned.

```
SELECT c.c_custkey, c.c_name, o.o_orderstatus, o.o_orderpriority
  FROM customer_clone c JOIN orders_clone o ON c.c_custkey=o.o_custkey;
```

Since the data cache was empty, performance of the JOIN on the clones should be very close to the performance on the tables that were created with CTAS.

3.3 Join Tables Using a Share

3.3.1 Run a query that joins two tables from SNOWFLAKE_SAMPLE_DATA.

```
SELECT c.c_custkey, c.c_name, o.o_orderstatus, o.o_orderpriority
  FROM snowflake_sample_data.tpch_sf100.customer c JOIN
     snowflake_sample_data.tpch_sf100.orders o ON c.c_custkey=o.o_custkey;
```

The SNOWFLAKE_SAMPLE_DATA database is a shared database: the data is actually located on the Snowflake cloud services layer rather than your local storage. This database is shared to all customers.

3.3.2 Record how long the query took, and how many rows were returned.

How does the performance compare now? The query on the local databases and the shared database should run in essentially the same amount of time. Since these tables are fairly small, there may be slight differences due to overhead.

3.3.3 Suspend your virtual warehouse to clear the data cache.

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

3.3.4 JOIN using a local table and a shared table.

```
SELECT c.c_custkey, c.c_name, o.o_orderstatus, o.o_orderpriority
  FROM snowflake_sample_data.tpch_sf100.customer c JOIN
     [login]_db.myschema.orders_ctas o ON c.c_custkey=o.o_custkey;
```

3.3.5 Check the execution time and number of rows returned. The performance should be comparable to the other JOINs you performed.

3.3.6 Turn the Query Result Cache back on so it is available for future labs.

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

3.3.7 Clean up the objects created for this lab.

```
DROP TABLE customer_ctas;
DROP TABLE customer_clone;
DROP TABLE orders_ctas;
DROP TABLE orders_clone;
```

4 Explore Account Security

This lab will take approximately *10 minutes* to complete.

4.1 Network Security

In this task, you will create a network policy and apply it to a user to see the effect.

4.1.1 Open a worksheet and use the role SECURITYADMIN.

4.1.2 Enter the following to create a network policy that allows access from only a single IP address:

```
CREATE NETWORK POLICY [login]_policy ALLOWED_IP_LIST=('12.13.14.15');
```


4.1.3 Describe the policy to verify that it was set correctly.

```
DESCRIBE NETWORK POLICY [login]_policy;
```

4.1.4 Now, create a new user:

```
CREATE USER [login]_testuser PASSWORD = 'Password@1' MUST_CHANGE_PASSWORD = FALSE;
```

4.1.5 Log out of the Snowflake account, and log back in as the user `[login]_testuser` you just created. This is just to verify that the new user can log in. After you have done that, log back out and log in as yourself `[login]` .

4.1.6 Now apply the network policy you created earlier to your test user:

```
ALTER USER [login]_testuser SET NETWORK_POLICY = [login]_policy;
```

4.1.7 Log out of the Snowflake account, and try to log in as your test user `[login]_testuser` . You will not be able to. Then log in as yourself `[login]` . Since the network policy has only been applied to the test user, it does not impact your ability to access the Snowflake account.

4.1.8 Drop the policy and user you created:

```
DROP USER [login]_testuser;
DROP NETWORK POLICY [login]_policy;
```

4.2 Account-Level Parameters

In this lab, you will explore parameters that can be set at various levels, and how parameters are affected by changes at a higher level.

4.2.1 In your worksheet, set your context to use the training role, your database, and the public schema:

```
USE ROLE training_role;
CREATE DATABASE IF NOT EXISTS [login]_db;
USE [login]_db.public;
```

4.2.2 Run the following command to see all the parameters that are available at the account level:

```
SHOW PARAMETERS FOR ACCOUNT;
```

Review the available parameters and their default values.

4.2.3 Check the default time travel retention time that is set at the account level:

```
SHOW PARAMETERS LIKE 'DATA_RETENTION_TIME_IN_DAYS' IN ACCOUNT ;
```

4.2.4 Create two new tables - one with the default retention time, and another with a different retention time:

```
CREATE OR REPLACE TABLE tt_default (col1 INT);

CREATE OR REPLACE TABLE tt_set30 (col2 INT)
DATA_RETENTION_TIME_IN_DAYS=30;
```

4.2.5 Verify the retention time for the two tables:

```
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_default;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_set30;
```

4.2.6 Change the retention time on your schema:

```
ALTER SCHEMA [login]_db.public SET DATA_RETENTION_TIME_IN_DAYS = 10;
```

4.2.7 Verify the retention time for the two tables:

```
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_default;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_set30;
```

You will see that the retention time on tt_set30, which was set to a specific value, did not change. The retention time on tt_default takes its value from the closest enclosing object, which is the schema.

4.2.8 Change the retention time for the database:

```
ALTER DATABASE [login]_db SET DATA_RETENTION_TIME_IN_DAYS = 2;
```

4.2.9 Verify the retention time for the schema, and the two tables:

```
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR SCHEMA public;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_default;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_set30;
```

You should see that all of the retention times are unaffected by this change - because tt_set30 has a specific value set and the schema, which also has a specific value set, passes its setting to tt_default. This is because the schema is tt_default's closest enclosing object that has the value set.

4.2.10 Reset the retention time for the schema back to the default:

```
ALTER SCHEMA [login]_db.public UNSET DATA_RETENTION_TIME_IN_DAYS;
```

Using UNSET ensures the schema will inherit its data retention time from its enclosing object. If you had set the data retention time to 1, it would use that value regardless of any changes made at the database or account level.

4.2.11 Verify the retention time for the schema, and the two tables:

```
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR SCHEMA public;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_default;
SHOW PARAMETERS LIKE 'DATA_RETENTION%' FOR TABLE tt_set30;
```

Since the schema and table tt_default do not have specific values set, they will now take on the value set by the closest enclosing object that has a set value (the database).

4.2.12 Drop your test tables, and reset the retention time for the database back to the default value:

```
DROP TABLE tt_default;
DROP TABLE tt_set30;
ALTER DATABASE [login]_db UNSET DATA_RETENTION_TIME_IN_DAYS;
```

5 Explore Users, Roles, and Privileges

This lab will take approximately *25 minutes* to complete.

This exercise will demonstrate the separation of functional (user) roles and object roles. In this lab you will create four functional roles and four object roles, to create the hierarchy shown here:

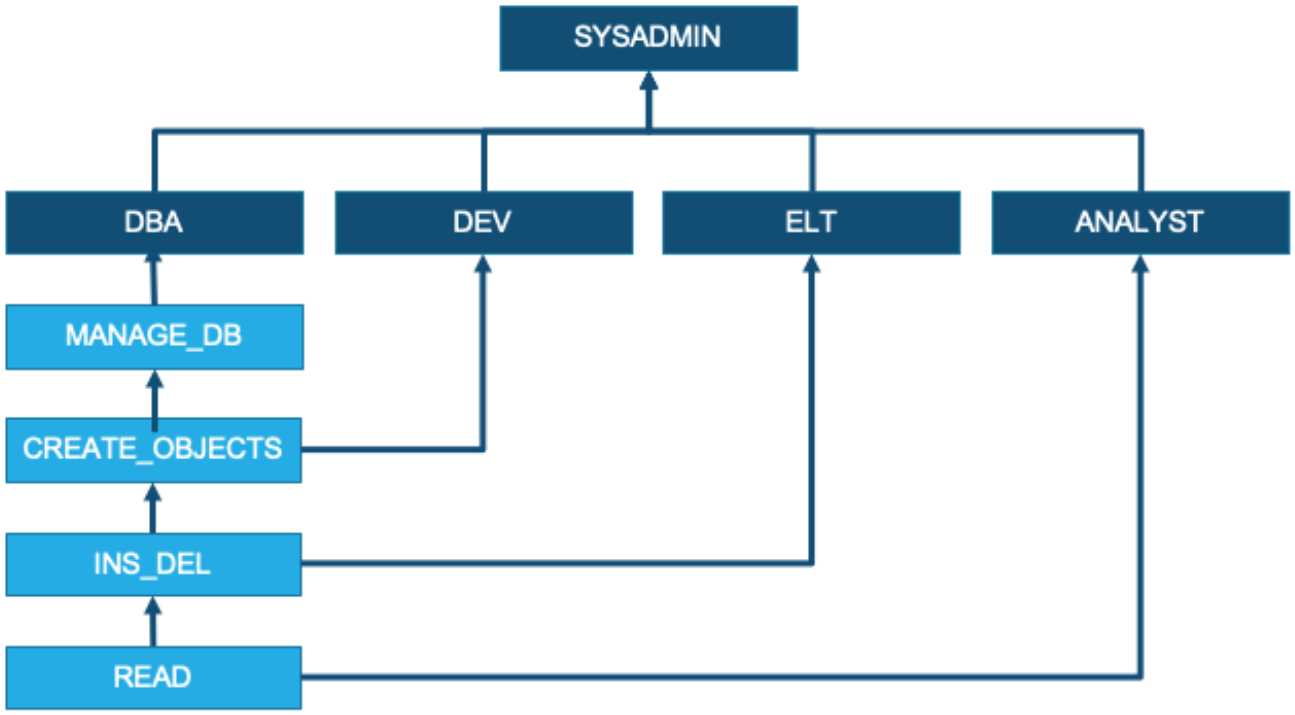

Figure 2: Role Diagram

5.1 Create and Configure Roles

5.1.1 Start by creating a database and schema you will use for this lab.

```
USE ROLE SYSADMIN;

CREATE DATABASE [login]_prod;
CREATE SCHEMA [login]_prod.sales;
```

5.1.2 Create the functional roles.

```
USE ROLE SECURITYADMIN;

CREATE ROLE [login]_dba;
CREATE ROLE [login]_dev;
CREATE ROLE [login]_elt;
CREATE ROLE [login]_analyst;
```

5.1.3 Grant each of these roles to SYSADMIN, to create the first part of the hierarchy.

```
GRANT ROLE [login]_dba TO ROLE SYSADMIN;
GRANT ROLE [login]_dev TO ROLE SYSADMIN;
```



```
GRANT ROLE [login]_elt TO ROLE SYSADMIN;
GRANT ROLE [login]_analyst TO ROLE SYSADMIN;
```

5.1.4 Assign your user to each of these roles.

In an actual deployment, each role would probably be granted to multiple users, and each role would have a different set of users.

```
GRANT ROLE [login]_dba
           ,[login]_dev
           ,[login]_elt
           ,[login]_analyst TO USER [login];
```

5.1.5 Create the object roles, which will be assigned privileges to access objects.

```
CREATE ROLE [login]_read;
CREATE ROLE [login]_ins_del;
CREATE ROLE [login]_create_objects;
CREATE ROLE [login]_manage_db;
```

5.1.6 Link the object roles together so privileges flow up through those roles.

```
GRANT ROLE [login]_read TO ROLE [login]_ins_del;
GRANT ROLE [login]_ins_del TO ROLE [login]_create_objects;
GRANT ROLE [login]_create_objects TO ROLE [login]_manage_db;
```

5.1.7 Grant the object roles to the appropriate functional roles.

Note that each object role is granted to a functional role (to define what that functional role can do), and to another object role (to build the hierarchy between object roles).

```
GRANT ROLE [login]_read TO ROLE [login]_analyst;
GRANT ROLE [login]_ins_del TO ROLE [login]_elt;
GRANT ROLE [login]_create_objects TO ROLE [login]_dev;
GRANT ROLE [login]_manage_db TO ROLE [login]_dba;
```

5.1.8 Grant USAGE Privileges to Object Roles

```
USE ROLE SYSADMIN;

GRANT USAGE ON DATABASE [login]_prod to ROLE [login]_read;

GRANT USAGE ON ALL SCHEMAS IN DATABASE [login]_prod to ROLE [login]_read;
GRANT USAGE ON FUTURE SCHEMAS IN DATABASE [login]_prod to ROLE [login]_read;
```


Because the READ role's privileges are rolled up the hierarchy, all object roles now have USAGE privileges on the database [login]_PROD, the schema SALES, and all future schemas in that database.

5.1.9 Grant object privileges to the object roles.

Keep in mind that privileges granted to one role will roll up to the parent roles - so you only have to grant privileges at the lowest level you want them to be available. Also, there are not yet any objects inside the schema SALES, so you only have to grant privileges on future objects.

```
GRANT SELECT ON FUTURE TABLES IN DATABASE [login]_prod TO ROLE [login]_read;
GRANT SELECT ON FUTURE VIEWS IN DATABASE [login]_prod TO ROLE [login]_read;
```

5.1.10 Grant privileges to the INS_DEL role.

```
GRANT INSERT, UPDATE, DELETE ON FUTURE TABLES IN DATABASE [login]_prod
  TO ROLE [login]_ins_del;
```

5.1.11 Grant privileges to the CREATE_OBJECTS role.

Since a schema already exists, you will need to grant privileges on the current schema (SALES), and any future schemas you want the role to have access to.

```
GRANT CREATE TABLE
     ,CREATE VIEW
     ,CREATE FILE FORMAT
     ,CREATE STAGE
     ,CREATE PIPE
     ,CREATE SEQUENCE
     ,CREATE FUNCTION
     ,CREATE PROCEDURE
  ON ALL SCHEMAS IN DATABASE [login]_prod TO ROLE [login]_create_objects;

GRANT CREATE TABLE
     ,CREATE VIEW
     ,CREATE FILE FORMAT
     ,CREATE STAGE
     ,CREATE PIPE
     ,CREATE SEQUENCE
     ,CREATE FUNCTION
     ,CREATE PROCEDURE
  ON FUTURE SCHEMAS IN DATABASE [login]_prod TO ROLE [login]_create_objects;
```

5.1.12 Grant privileges to the MANAGE_DB role.

```
GRANT MODIFY
     ,MONITOR
     ,CREATE SCHEMA
```



```
ON DATABASE [login]_prod
TO ROLE [login]_manage_db;
```

5.2 Verify the Current Hierarchy

5.2.1 Verify that you are assigned to the functional roles, and that the functional roles have been granted to the role SYSADMIN.

```
USE ROLE SECURITYADMIN;

SHOW GRANTS OF ROLE [login]_dba;
SHOW GRANTS OF ROLE [login]_analyst;
SHOW GRANTS OF ROLE [login]_elt;
SHOW GRANTS OF ROLE [login]_dev;
```

5.2.2 Verify that you are NOT assigned to the object roles.

In your design, users are only assigned to the functional roles. Each object role should be assigned to one or more functional roles.

```
SHOW GRANTS OF ROLE [login]_read;
SHOW GRANTS OF ROLE [login]_ins_del;
SHOW GRANTS OF ROLE [login]_create_objects;
SHOW GRANTS OF ROLE [login]_manage_db;
```

5.2.3 Verify the grants on the [login]_PROD database.

```
SHOW GRANTS ON DATABASE [login]_prod;
```

5.3 Create and Grant Privileges on Warehouses

5.3.1 Create the warehouses.

```
USE ROLE SYSADMIN;

CREATE WAREHOUSE IF NOT EXISTS [login]_elt_wh
  WAREHOUSE_SIZE=XSmall
  INITIALLY_SUSPENDED=TRUE
  AUTO_SUSPEND=60;

CREATE WAREHOUSE IF NOT EXISTS [login]_small_query_wh
   WAREHOUSE_SIZE=Small
   INITIALLY_SUSPENDED=TRUE
   AUTO_SUSPEND=600;

CREATE WAREHOUSE IF NOT EXISTS [login]_large_query_wh
```



```
   WAREHOUSE_SIZE=Large
   INITIALLY_SUSPENDED=TRUE
   AUTO_SUSPEND=1200;

CREATE WAREHOUSE IF NOT EXISTS [login]_mc_wh
   WAREHOUSE_SIZE=Medium
   INITIALLY_SUSPENDED=TRUE
   MIN_CLUSTER_COUNT=1
   MAX_CLUSTER_COUNT=8
   AUTO_SUSPEND=300;
```

5.3.2 Grant privileges to the appropriate roles.

```
GRANT USAGE ON WAREHOUSE [login]_elt_wh TO ROLE [login]_ins_del;
GRANT USAGE ON WAREHOUSE [login]_small_query_wh TO ROLE [login]_read;
GRANT USAGE ON WAREHOUSE [login]_large_query_wh TO ROLE [login]_read;
GRANT USAGE ON WAREHOUSE [login]_mc_wh TO ROLE [login]_read;
```

5.4 Explore and Test Ownership and Privileges

Ownership is a privilege that is automatically granted to the role that creates an object. Ownership can be transferred to another role, but only by the current owner or the ACCOUNTADMIN role.

5.4.1 Check ownership of the SALES schema.

```
SHOW SCHEMAS IN DATABASE [login]_prod;
```

You should see that the PUBLIC and SALES schemas are owned by SYSADMIN. INFORMATION_SCHEMA does not have an owner, which is expected.

5.4.2 Transfer ownership of the SALES schema to the [login]_MANAGE_DB role.

The company wants centralized management, and since `[login]_manage_db` can create schemas, it should own all schemas.

```
USE ROLE SYSADMIN;
GRANT OWNERSHIP ON SCHEMA [login]_prod.sales TO ROLE [login]_manage_db COPY
   CURRENT GRANTS;
```

The COPY CURRENT GRANTS clause means that any privileges on the schema will be preserved.

5.4.3 Check ownership of the SALES schema.

```
SHOW SCHEMAS LIKE 'sales';
```


5.4.4 Make SALES a MANAGED ACCESS schema.

```
ALTER SCHEMA [login]_prod.sales ENABLE MANAGED ACCESS;
```

When a role creates an object, that role owns it. Normally, the owner of an object can do anything with it, including grant privileges on that object to other roles. The MANAGED ACCESS clause means that only the **schema** owner can grant privileges on the objects inside the schema, regardless of who owns them. This centralizes the control of object privileges.

5.4.5 Test the DBA role.

Once you have set up your role-based access control, you should test it to verify that it works as expected.

This is the most powerful custom functional role in your hierarchy. It should be able to create schemas and objects, and perform any operation on those objects. It should also be able to grant privileges on any objects in the schemas it owns.

```
USE ROLE [login]_dba;
USE DATABASE [login]_prod;
USE WAREHOUSE [login]_small_query_wh;

CREATE SCHEMA finance;
USE SCHEMA finance;

CREATE TABLE dba_fin_tbl (c1 INT);
INSERT INTO dba_fin_tbl VALUES (1), (2), (3), (4), (5), (6);
SELECT * FROM dba_fin_tbl;
DELETE FROM dba_fin_tbl WHERE c1=6;
SELECT * FROM dba_fin_tbl;

USE SCHEMA sales;
CREATE TABLE partners (acct INT, company_name VARCHAR, sales_2020 NUMBER(10,2));
INSERT INTO partners VALUES (1004, 'Expedia', 1234548.12),
                            (1332, 'Kayak', 987256.95),
                            (1447, 'Priceline', 245334.78);
```

5.4.6 Test the DEV role.

The DEV role was granted privileges to create objects. It also inherited SELECT, INSERT, UPDATE, DELETE, and TRUNCATE from the roles below it. It was also given usage rights on future schemas. You want to verify that this role can use a warehouse and create and manipulate objects. You also want to verify that this role cannot create schemas, but it can create objects in schemas that the DBA created.

Run the following statements to determine if the DEV role was configured correctly, based on what it was designed to do.

```
USE ROLE [login]_dev;
USE [login]_prod.sales;
USE WAREHOUSE [login]_small_query_wh;
```



```
CREATE TABLE dev_sales_tbl (c1 INT, c2 VARCHAR(1));
INSERT INTO dev_sales_tbl VALUES (1, 'A'), (2, 'B'), (3, 'C'), (4,'D'), (5, 'E');
SELECT * FROM dev_sales_tbl;

CREATE VIEW dev_sales_view AS
  SELECT * FROM dev_sales_tbl WHERE c1=3;

SELECT * FROM dev_sales_view;

SELECT * FROM partners;

USE SCHEMA finance;

TRUNCATE TABLE dba_fin_tbl;

INSERT INTO dba_fin_tbl VALUES (2), (4), (6), (8), (10);

SELECT f.c1, s.c2 FROM dba_fin_tbl f
   JOIN sales.dev_sales_tbl s
   ON f.c1 = s.c1;

CREATE SCHEMA dev_schema;
```

The `CREATE SCHEMA` and `TRUNCATE TABLE` commands should fail, but all the others should work.

5.4.7 Test the ELT role.

The ELT role cannot create objects, but it can SELECT, INSERT, UPDATE, and DELETE. Test the capabilities of the ELT role. Some of these commands will work, and others will fail.

Run the following statements to determine if the ELT role was configured correctly, based on what it was designed to do.

```
USE ROLE [login]_elt;
USE DATABASE [login]_prod;
USE WAREHOUSE [login]_elt_wh;

SELECT * FROM sales.dev_sales_tbl;
INSERT INTO sales.dev_sales_tbl VALUES (6, 'F'), (7, 'G'), (8, 'H'), (9, 'I');
SELECT * FROM sales.dev_sales_tbl;
DELETE FROM sales.dev_sales_tbl WHERE c2='H';
SELECT * FROM sales.dev_sales_tbl;

SELECT * FROM finance.dba_fin_tbl;

TRUNCATE finance.dba_fin_tbl;

DELETE FROM sales.partners WHERE sales_2020 < 100000;
```

The ELT role was configured to `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `TRUNCATE` the table.

5.4.8 Test the ANALYST role.

The ANALYST role can use a warehouse and select data, but has not other privileges. Verify that this is the case. Again, some of the command below will succeed and others will fail. Is this role set up correctly?

```
USE ROLE [login]_analyst;
USE DATABASE [login]_prod;
USE WAREHOUSE [login]_elt_wh;
USE WAREHOUSE [login]_large_query_wh;

CREATE SCHEMA test;

CREATE TABLE sales.analyst_tbl (c1 INT);

USE SCHEMA finance;

INSERT INTO dba_fin_tbl VALUES (11), (12), (13);

SELECT * FROM dba_fin_tbl;

SELECT f.c1, s.c2 FROM dba_fin_tbl f
   JOIN sales.dev_sales_tbl s
   ON f.c1 = s.c1;

SELECT * FROM sales.partners
   WHERE company_name = 'Expedia';
```

5.4.9 Grant the ability to create users to the DBAs.

The company has decided that the DBA should be more powerful. DBAs should be able to create users and assign them to roles, and should also be able to operate and monitor warehouses.

```
USE ROLE SECURITYADMIN;

GRANT CREATE USER ON ACCOUNT TO ROLE [login]_manage_db;
```

Note that we grant these privileges on the object role that rolls up to the functional (DBA) role, not to the functional role itself.

5.4.10 Also give DBAs the ability to assign users to roles.

```
GRANT MANAGE GRANTS ON ACCOUNT
   TO ROLE [login]_manage_db;
```

5.4.11 Finally, allow the DBA to operate and monitor warehouses.

```
USE ROLE SYSADMIN;

GRANT OPERATE, MONITOR ON WAREHOUSE [login]_elt_wh TO ROLE [login]_manage_db;
```



```
GRANT OPERATE, MONITOR ON WAREHOUSE [login]_mc_wh TO ROLE [login]_manage_db;
GRANT OPERATE, MONITOR ON WAREHOUSE [login]_small_query_wh TO ROLE
   [login]_manage_db;
GRANT OPERATE, MONITOR ON WAREHOUSE [login]_large_query_wh TO ROLE
   [login]_manage_db;
```

5.4.12 Test the new privileges.

The DBA should be able to create users, assign then to roles, and start or stop (operate) the warehouses. The DBA is not able to modify the warehouses.

```
USE ROLE [login]_dba;

CREATE USER [login]_test_user;
GRANT ROLE [login]_dba TO USER [login]_test_user;

ALTER WAREHOUSE [login]_elt_wh RESUME;
ALTER WAREHOUSE [login]_elt_wh SUSPEND;

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

Since the DBA cannot modify a warehouse an error is expected.

5.5 Explore and Testing Revoking Privileges

Revoking privileges may be needed to prevent a role, for example, from selecting from a privileged table. The company does not want the analysts selecting from the partners table.

5.5.1 Confirm that you can currently select from the table using the analyst role.

```
USE ROLE [login]_analyst;
USE WAREHOUSE [login]_small_query_wh;

SELECT * FROM [login]_prod.sales.partners;
```

5.5.2 Revoke privileges on the table.

```
USE ROLE [login]_dba;

REVOKE SELECT ON TABLE [login]_prod.sales.partners
   FROM ROLE [login]_read;
```

5.5.3 Confirm that you no longer can select from the table using the analyst role.


```
USE ROLE [login]_analyst;
USE [login]_prod.sales;
USE WAREHOUSE [login]_small_query_wh;

SELECT * FROM partners;
```

Since the role cannot select from a table an error is expected.

5.5.4 Verify that the analyst can still select from other tables in the SALES schema.

```
SELECT * FROM dev_sales_tbl;
```

5.5.5 See if any other roles higher up in the hierarchy can select from the partners table.

```
USE ROLE [login]_elt;
SELECT * FROM partners;

USE ROLE [login]_dev;
SELECT * FROM partners;

USE ROLE [login]_dba;
SELECT * FROM partners;
```

What happened, and why?

5.5.6 To try and find out, revoke privileges from another table and see which roles can still read it.

```
USE ROLE [login]_dba;
REVOKE SELECT ON TABLE [login]_prod.sales.dev_sales_tbl
   FROM ROLE [login]_read;

USE ROLE [login]_analyst;
SELECT * FROM [login]_prod.sales.dev_sales_tbl;

USE ROLE [login]_ELT;
SELECT * FROM [login]_prod.sales.dev_sales_tbl;

USE ROLE [login]_DEV;
SELECT * FROM [login]_prod.sales.dev_sales_tbl;

USE ROLE [login]_DBA;
SELECT * FROM [login]_prod.sales.dev_sales_tbl;
```

What were the results, and what does this tell you?

5.5.7 Run this command to help you figure it out:


```
SHOW TABLES;
```

In each case, the owner of the table could still read it, even after SELECT privileges were revoked from the READ role. You cannot revoke privileges from an object owner.

5.6 Clean Up

Run this series of commands all at once, to clean up the roles and objects that were created for this exercise.

5.6.1 Remove the roles, and the user that the DBA created.

```
USE ROLE SECURITYADMIN;

DROP ROLE [login]_dba;
DROP ROLE [login]_dev;
DROP ROLE [login]_elt;
DROP ROLE [login]_analyst;

DROP ROLE [login]_read;
DROP ROLE [login]_ins_del;
DROP ROLE [login]_create_objects;
DROP ROLE [login]_manage_db;

DROP USER [login]_test_user;
```

5.6.2 Remove the objects.

```
USE ROLE SYSADMIN;

DROP DATABASE [login]_prod;
```

Note that if you drop a database, all the schemas and objects within that database are automatically dropped as well. Warehouses are independent of databases, so those do need to be dropped separately.

```
DROP WAREHOUSE [login]_small_query_wh;
DROP WAREHOUSE [login]_large_query_wh;
DROP WAREHOUSE [login]_elt_wh;
DROP WAREHOUSE [login]_mc_wh;
```

6 Loading, Transforming and Validating Data

This lab will take approximately *45 minutes* to complete.

You will learn how to load data into Snowflake using external stages and file formats, and transform data upon load. Also you will use Snowflake's `VALIDATION_MODE` option on a `COPY` statement to demonstrate Snowflake's pre-load error detection mechanism.

6.1 Load Structured Data

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 (5) 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 in the *region.tbl* file there is a delimiter at the end of every line, which by default is interpreted as an additional column by the COPY INTO statement.

The files required for this lab are in an external stage.

6.1.1 Navigate to Worksheets and create a new worksheet. Name it "Data Movement".

6.1.2 Set the Worksheet contexts as follows:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_LOAD_WH
   WAREHOUSE_SIZE=XSmall
   INITIALLY_SUSPENDED=True
   AUTO_SUSPEND=300;
USE WAREHOUSE [login]_LOAD_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
USE DATABASE [login]_DB;
USE SCHEMA PUBLIC;
```

6.1.3 Create the staging tables by running the following statements

```
CREATE OR REPLACE TABLE REGION (
       R_REGIONKEY NUMBER(38,0) NOT NULL,
       R_NAME      VARCHAR(25)  NOT NULL,
       R_COMMENT   VARCHAR(152)
);
CREATE OR REPLACE TABLE NATION (
       N_NATIONKEY NUMBER(38,0) NOT NULL,
       N_NAME      VARCHAR(25)  NOT NULL,
       N_REGIONKEY NUMBER(38,0) NOT NULL,
       N_COMMENT   VARCHAR(152)
);
CREATE OR REPLACE TABLE SUPPLIER (
       S_SUPPKEY   NUMBER(38,0) NOT NULL,
       S_NAME      VARCHAR(25)  NOT NULL,
       S_ADDRESS   VARCHAR(40)  NOT NULL,
       S_NATIONKEY NUMBER(38,0) NOT NULL,
       S_PHONE     VARCHAR(15)  NOT NULL,
```



```
        S_ACCTBAL   NUMBER(12,2) NOT NULL,
        S_COMMENT   VARCHAR(101)
);
CREATE OR REPLACE TABLE PART (
        P_PARTKEY     NUMBER(38,0) NOT NULL,
        P_NAME        VARCHAR(55)  NOT NULL,
        P_MFGR        VARCHAR(25)  NOT NULL,
        P_BRAND       VARCHAR(10)  NOT NULL,
        P_TYPE        VARCHAR(25)  NOT NULL,
        P_SIZE        NUMBER(38,0) NOT NULL,
        P_CONTAINER   VARCHAR(10)  NOT NULL,
        P_RETAILPRICE NUMBER(12,2) NOT NULL,
        P_COMMENT     VARCHAR(23)
);
CREATE OR REPLACE TABLE PARTSUPP (
        PS_PARTKEY    NUMBER(38,0) NOT NULL,
        PS_SUPPKEY    NUMBER(38,0) NOT NULL,
        PS_AVAILQTY   NUMBER(38,0) NOT NULL,
        PS_SUPPLYCOST NUMBER(12,2) NOT NULL,
        PS_COMMENT    VARCHAR(199)
);
CREATE OR REPLACE TABLE CUSTOMER (
        C_CUSTKEY    NUMBER(38,0) NOT NULL,
        C_NAME       VARCHAR(25)  NOT NULL,
        C_ADDRESS    VARCHAR(40)  NOT NULL,
        C_NATIONKEY  NUMBER(38,0) NOT NULL,
        C_PHONE      VARCHAR(15)  NOT NULL,
        C_ACCTBAL    NUMBER(12,2) NOT NULL,
        C_MKTSEGMENT VARCHAR(10),
        C_COMMENT    VARCHAR(117)
);
CREATE OR REPLACE TABLE ORDERS (
        O_ORDERKEY      NUMBER(38,0) NOT NULL,
        O_CUSTKEY       NUMBER(38,0) NOT NULL,
        O_ORDERSTATUS   VARCHAR(1)   NOT NULL,
        O_TOTALPRICE    NUMBER(12,2) NOT NULL,
        O_ORDERDATE     DATE         NOT NULL,
        O_ORDERPRIORITY VARCHAR(15)  NOT NULL,
        O_CLERK         VARCHAR(15)  NOT NULL,
        O_SHIPPRIORITY  NUMBER(38,0) NOT NULL,
        O_COMMENT       VARCHAR(79)  NOT NULL
);
CREATE OR REPLACE TABLE LINEITEM (
        L_ORDERKEY      NUMBER(38,0) NOT NULL,
        L_PARTKEY       NUMBER(38,0) NOT NULL,
        L_SUPPKEY       NUMBER(38,0) NOT NULL,
        L_LINENUMBER    NUMBER(38,0) NOT NULL,
        L_QUANTITY      NUMBER(12,2) NOT NULL,
        L_EXTENDEDPRICE NUMBER(12,2) NOT NULL,
        L_DISCOUNT      NUMBER(12,2) NOT NULL,
        L_TAX           NUMBER(12,2) NOT NULL,
        L_RETURNFLAG    VARCHAR(1)   NOT NULL,
        L_LINESTATUS    VARCHAR(1)   NOT NULL,
        L_SHIPDATE      DATE         NOT NULL,
        L_COMMITDATE    DATE         NOT NULL,
        L_RECEIPTDATE   DATE         NOT NULL,
```



```
        L_SHIPINSTRUCT  VARCHAR(25)  NOT NULL,
        L_SHIPMODE      VARCHAR(10)  NOT NULL,
        L_COMMENT       VARCHAR(44)  NOT NULL
);
CREATE OR REPLACE TABLE COUNTRYGEO (
        CG_NATIONKEY NUMBER(38,0),
        CG_CAPITAL   VARCHAR(100),
        CG_LAT       NUMBER(20,10),
        CG_LON       NUMBER(20,10),
        CG_ALTITUDE  NUMBER(38,0)
);
```

6.1.4 Find the region.tbl file in the external stage with list and a regex pattern.

```
LIST @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/ pattern='.*region.*';
```

6.1.5 Load the data from the external stage to the REGION Table using the MYPIPEFORMAT file format.

```
DESCRIBE FILE FORMAT TRAINING_DB.TRAININGLAB.MYPIPEFORMAT;

COPY INTO REGION
  FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/
  FILES = ( 'region.tbl' )
  FILE_FORMAT = ( FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT );
```

The file formats required for the lab steps have been created and are all located in the TRAININGLAB schema.

6.1.6 Select and review the data in the REGION Table:

```
SELECT * FROM REGION;
```

6.1.7 Preview the REGION table in the WebUI using the sidebar.

First, click on your database [login]_DB in the navigator. Locate and click on the PUBLIC schema. In the list of tables, find the REGION table. Click the series of three ellipses to the right and select `Preview Data`:

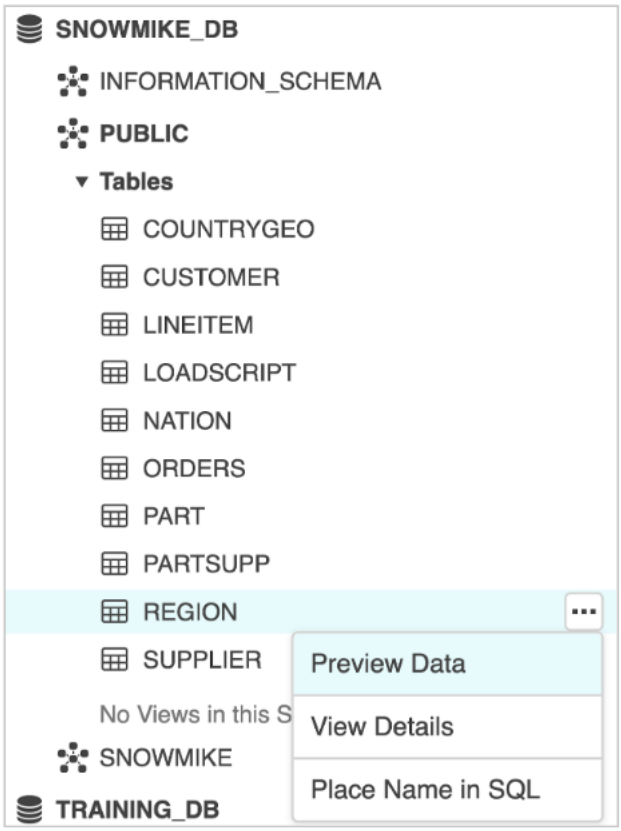

Figure 3: Select Preview Data

6.2 Loading Data and File Sizes

When loading data, file size matters. Snowflake recommends using files sizes of 100MB to 250MB of compressed data for both bulk loading using COPY and for streaming using Snowpipe. Both MACOS and Linux support a file splitting utility.

6.2.1 Set Context:

```
USE ROLE training_role;
USE SCHEMA [login]_db.public;
USE WAREHOUSE [login]_load_wh;
```

6.2.2 Create a named stage:

```
CREATE  OR REPLACE TEMPORARY STAGE [login]_STAGE;
```

6.2.3 Change your warehouse size to small:


```
ALTER WAREHOUSE [login]_load_wh SET WAREHOUSE_SIZE = SMALL;
```

6.2.4 Download a file from the Citibike table to the stage you created:

```
COPY INTO @[login]_STAGE/citibike/singlefile/citibike.tbl
  FROM citibike.schema1.trips
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYPIPEFORMAT)
  SINGLE=TRUE
  MAX_FILE_SIZE=5368709120;
```

You will receive an error message - this is because the file is too large to be downloaded as a single file.

6.2.5 Confirm the error:

```
ls @[login]_STAGE/citibike/singlefile;
```

6.2.6 Re-run the command to limit the amount of data unloaded:

```
COPY INTO @[login]_STAGE/citibike/singlefile/citibike.tbl
  FROM (SELECT * FROM citibike.schema1.trips LIMIT 20000000)
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYPIPEFORMAT)
  SINGLE=TRUE
  MAX_FILE_SIZE=5368709120;
```

6.2.7 List the file on the stage:

```
ls @[login]_STAGE/citibike/singlefile;
```

6.2.8 Now resize your warehouse to LARGE and unload the data to the stage without using the SINGLE option:

```
ALTER WAREHOUSE [login]_load_wh SET WAREHOUSE_SIZE = LARGE;

COPY INTO @[login]_STAGE/citibike/multiplefiles/citibike_
  FROM (SELECT * FROM citibike.schema1.trips)
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYPIPEFORMAT)
  SINGLE=FALSE;

ls @[login]_STAGE/citibike/multiplefiles;

ALTER WAREHOUSE [login]_load_wh set warehouse_size = SMALL;
```


6.3 Load Semi-Structured Data

This exercise will load tables from text files that are in an external stage. You will load text files from an external stage using the Web UI.

6.3.1 Navigate to Databases in the top ribbon.

6.3.2 Select the TRAINING_DB Database.

6.3.3 Navigate to the Stages area.

6.3.4 Confirm you see the ED_STAGE Stage in the TRAINING_DB.TRAININGLAB Schema.

Take note of its location. This external stage points to an AWS S3 bucket.

6.3.5 Navigate to Worksheets.

6.3.6 List the files in the TRAINING_DB.TRAININGLAB.ED_STAGE/coredata stage:

```
ls @TRAINING_DB.TRAININGLAB.ED_STAGE/coredata/TCPH/TCPH_SF100;

--Set a variable to hold the query id of the ls command
SET sf100 = LAST_QUERY_ID();
```

There are many files to load for some of the larger tables. For these larger tables, you will get better performance using more cores (a larger virtual warehouse). Therefore, for this load exercise you will alter the size of the virtual warehouse to an X-Large. In a real-world application, you would create a load cluster sized to the number of files that you plan to load.

6.3.7 Alter the [login]_load_wh and increase its size:

```
ALTER WAREHOUSE [login]_LOAD_WH SET WAREHOUSE_SIZE='X-LARGE';
```

6.3.8 Query all of the unique directories that follow TCP_SF100 in the stage.

```
SELECT DISTINCT REGEXP_SUBSTR(  --
   https://docs.snowflake.net/manuals/sql-reference/functions/regexp_substr.html
                    "name"     -- column name that contains the string
                    ,'.*TCPH_SF100\/(.*)\/'  --regular expression string, to see
                       how it works you can visit https://regex101.com/r/r6pX3O/1
```



```
                    ,1          --start from the beginning of the string
                    ,1          --find the first occurrance match
                    ,'e'        --extract the sub-matches
                    ,1          --return the first sub match
                  ) AS  DIRECTORY_NAMES
   FROM TABLE(RESULT_SCAN($sf100));
```

6.3.9 Load data into the `CUSTOMER` table.

```
COPY INTO [login]_DB.PUBLIC.CUSTOMER
  FROM @TRAINING_DB.TRAININGLAB.ED_STAGE
  PATTERN='.*/CUSTOMER/.*tbl'
  FILE_FORMAT = (FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT)
  ON_ERROR = 'CONTINUE';
```

6.3.10 Load the remaining tables.

Using the above `COPY INTO` command as a template, run additional `COPY INTO` statements to load each table in your `[login]_DB.PUBLIC` schema.

Remaining tables:

- PARTSUPP
- ORDERS
- LINEITEM
- NATION
- SUPPLIER
- PART

6.3.11 Count the number of rows from the newly populated tables:

```
SELECT TABLE_NAME, ROW_COUNT
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = CURRENT_SCHEMA();
```

6.3.12 **BONUS** Try re-writing the above query to also get the number of files in each directory.

6.4 Load Semi-Structured Parquet Data

This exercise will load a Parquet data file using different methods.

6.4.1 Empty the REGION Table in the PUBLIC schema of your [login]_DB:

```
TRUNCATE TABLE REGION;
```


6.4.2 Confirm that the region.parquet file is in the External Stage:

```
LIST @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files
    PATTERN = '.*region.*';

LIST @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files
    PATTERN = '.*region.*parquet$';
```

6.4.3 Create the file format for the Parquet file in the current schema:

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

SELECT *
FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/region.parquet
(FILE_FORMAT => MYPARQUETFORMAT);
```

6.4.4 Query the data in the region.parquet file from the external stage:

```
SELECT
      $1,
      $1:_COL_0::number,
      $1:_COL_1::varchar,
      $1:_COL_2::varchar
 FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/region.parquet
 (FILE_FORMAT => MYPARQUETFORMAT);
```

6.4.5 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 @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/
     )
FILES = ('region.parquet')
FILE_FORMAT = (FORMAT_NAME = MYPARQUETFORMAT);
```

6.4.6 View the data:

```
SELECT * FROM REGION;
```


6.5 Load Semi-Structured JSON Data

This exercise will load a JSON data file.

6.5.1 Confirm that the countrygeo.json file is in the External Stage:

```
LIST @TRAINING_DB.TRAININGLAB.ED_STAGE PATTERN = '.*countrygeo.*';
```

6.5.2 Query the file directly from the stage:

```
SELECT *
FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/countrygeo.json
(FILE_FORMAT => 'TRAINING_DB.TRAININGLAB.MYJSONFORMAT');
```

6.5.3 Load the COUNTRYGEO Table from the countrygeo.json file:

```
CREATE OR REPLACE TABLE COUNTRYGEO (CG_V variant);

COPY INTO COUNTRYGEO (CG_V)
FROM (SELECT $1
      FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/countrygeo.json )
      FILE_FORMAT = ( FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYJSONFORMAT )
      ON_ERROR = 'continue';
```

6.5.4 View the data:

```
SELECT * FROM COUNTRYGEO;
```

6.6 Load Fixed Format Data

Loading fixed format data takes advantage of Snowflake's ability to transform data upon load. The approach is to load the data into a VARCHAR or STRING column and use Snowflake functions to transform the data and load it.

6.6.1 Set Context

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


6.6.2 Create the target NATION table from TCPH data.

```
CREATE TABLE nation_tbl LIKE training_db.traininglab.nation;
```

6.6.3 Validate that the source file exists on the stage.

```
LS @training_db.traininglab.ed_stage/coredata/TCPH/FIXFORMAT;
```

6.6.4 Create a File Format object.

```
CREATE FILE FORMAT [login]_FIXED TYPE = 'CSV'
                   COMPRESSION = 'AUTO'
                   FIELD_DELIMITER = 'NONE'
                   RECORD_DELIMITER = '\n'
                   FIELD_OPTIONALLY_ENCLOSED_BY = 'NONE'
                   ESCAPE = 'NONE' ;
```

6.6.5 Use the copy statement to load the data from Stage

```
COPY INTO nation_tbl
  FROM (SELECT CAST(SUBSTR($1,1,2) AS NUMBER)
               ,SUBSTR($1,3,12)
               ,CAST(SUBSTR($1,19,1) AS NUMBER)
               ,SUBSTR($1,20,114)
         FROM '@training_db.traininglab.ed_stage/coredata/TCPH/FIXFORMAT'
  )
           FILE_FORMAT=(FORMAT_NAME=[login]_FIXED);
```

6.6.6 Query the loaded data.

```
SELECT * FROM nation_tbl;
```

Sample data:

```
1 ARGENTINA      1al foxes promise slyly according to the regular accounts. bold
   requests alon
2 BRAZIL         1y alongside of the pending deposits. carefully special packages
   are about the ironic forges. slyly special
3 CANADA         1eas hang ironic, silent packages. slyly regular packages are
   furiously over the tithes. fluffily bold
4 EGYPT          4y above the carefully unusual theodolites. final dugouts are
   quickly across the furiously regular d
```


6.7 Detect File Format Problems with VALIDATION_MODE

Use Snowflake's `VALIDATION_MODE` option on a `COPY` statement to demonstrate Snowflake's pre-load error detection mechanism.

6.7.1 Set the following context:

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

6.7.2 Create (or replace from previous labs) the `REGION` table to get ready to load an empty table:

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

6.7.3 Run a COPY command in validation mode against `region_bad_1.tbl`, and identify the issue that will cause the load to fail:

```
COPY INTO region
FROM @training_db.traininglab.ed_stage/load/lab_files/
  FILES = ( 'region_bad_1.tbl' )
  FILE_FORMAT = ( FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT )
  VALIDATION_MODE = RETURN_ALL_ERRORS;
```

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 in the first column, which should contain a number, the character is `x`.

6.7.4 Run a COPY command in validation mode against `region_bad_2.tbl` and identify the issue that will cause the load to fail:

```
COPY INTO region FROM @training_db.traininglab.ed_stage/load/lab_files/
  FILES = ('region_bad_2.tbl')
  FILE_FORMAT = (FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT)
  VALIDATION_MODE = RETURN_ALL_ERRORS;
```

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

6.8 Load Data with ON_ERROR set to CONTINUE

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

6.8.1 Run a COPY command with the ON_ERROR parameter set to CONTINUE:

```
COPY INTO REGION FROM @training_db.traininglab.ed_stage/load/lab_files/
  FILES = ( 'region_bad_1.tbl' )
  FILE_FORMAT = ( FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT )
  ON_ERROR = CONTINUE;
```

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

6.8.2 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.

```
SELECT * FROM region;
```

You should see that the row with REGIONKEY 1 is missing.

6.8.3 Truncate the `REGION` table:

```
TRUNCATE TABLE region;
```

6.8.4 Run a COPY command with the ON_ERROR parameter set to CONTINUE against `region_bad_2.tbl`:

```
COPY INTO region FROM @training_db.traininglab.ed_stage/load/lab_files/
  FILES = ('region_bad_2.tbl')
  FILE_FORMAT = (FORMAT_NAME = TRAINING_DB.TRAININGLAB.MYPIPEFORMAT)
  ON_ERROR = CONTINUE;
```

6.8.5 View the data that was loaded

Confirm that all rows were loaded except the row the `VALIDATION_MODE` against this file stated would not load (row 2).

```
SELECT * FROM region;
```

This time, REGIONKEY 2 is missing:

6.9 Reload the Region Table with Clean Data

6.9.1 Truncate the `REGION` table:

```
TRUNCATE TABLE region;
```

6.9.2 Validate that you have data in the table stage for your region table:

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

6.9.3 Load clean data to the `REGION` table:

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

6.9.4 View the data that was loaded and confirm that all 5 rows were loaded.

```
SELECT * FROM region;
```

6.9.5 Navigate to **[Databases]** in the top ribbon.

6.9.6 Select **[login]**_DB Database and confirm that the `REGION` table has data loaded.

7 Using Snowflake Pipes

This lab will take approximately *15 minutes* to complete.

There are two (2) types of Snowpipes, namely *basic pipes* and *continuous load pipes*. In this exercise you will work with basic pipes which you must manually refresh. A continuous load pipe will be refreshed automatically either from an API or the cloud provider's notification system.

7.1 Set Up a Basic Snowpipe

7.1.1 Set your context.

```
USE ROLE training_role;
CREATE WAREHOUSE IF NOT EXISTS [login]_query_wh
   WAREHOUSE_SIZE=XSmall
   INITIALLY_SUSPENDED=True
   AUTO_SUSPEND=300;
```



```
USE WAREHOUSE [login]_query_wh;
USE DATABASE [login]_db;
CREATE SCHEMA citibike;
USE SCHEMA citibike;
```

7.1.2 Create a Citibike trips table.

```
CREATE OR REPLACE TABLE trips (
    tripduration INTEGER
   ,starttime TIMESTAMP
   ,stoptime TIMESTAMP
   ,start_station_id INTEGER
   ,start_station_name STRING
   ,start_station_latitude FLOAT
   ,start_station_longitude FLOAT
   ,end_station_id INTEGER
   ,end_station_name STRING
   ,end_station_latitude FLOAT
   ,end_station_longitude FLOAT
   ,bikeid INTEGER
   ,membership_type STRING
   ,usertype STRING
   ,birth_year INTEGER
   ,gender INTEGER);
```

7.1.3 Create a pipe to load the Citibike Trips table.

```
CREATE OR REPLACE PIPE [login]_db.citibike.trips_pipe  AS
  COPY INTO [login]_db.citibike.trips
    FROM (SELECT *
    FROM @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/[login])
    FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYGZIPPIPEFORMAT);
SHOW PIPES;
```

7.2 Load the Trips Table

7.2.1 Unload data from the existing CITIBIKE database onto the stage.

```
COPY INTO @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/[login]/citibike1_
  FROM  (SELECT *  FROM CITIBIKE.SCHEMA1.TRIPS SAMPLE(10 ROWS))
   FILE_FORMAT = (FORMAT_NAME = training_db.traininglab.MYGZIPPIPEFORMAT);
```

7.2.2 List files on the stage.

```
LS @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/[login];
```

You should see new files loaded onto the stage location specified in the COPY statement.

7.2.3 Refresh the stage and check the data after loading it:

```
ALTER PIPE [login]_db.citibike.trips_pipe REFRESH;

SELECT system$pipe_status('[login]_db.citibike.trips_pipe');

SELECT * FROM trips;
```

7.2.4 Stage more data and refresh the pipe.

```
COPY INTO @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/[login]/citibike2_
  FROM  (SELECT *  FROM CITIBIKE.SCHEMA1.TRIPS SAMPLE(10 ROWS))
   FILE_FORMAT = (FORMAT_NAME = training_db.traininglab.MYGZIPPIPEFORMAT)
   OVERWRITE = TRUE;

ALTER PIPE [login]_db.citibike.trips_pipe REFRESH;
```

7.2.5 List files on the stage.

```
LS @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/[login];
```

7.2.6 Query the table to see the freshly loaded data (this might take a few minutes).

```
SELECT * FROM trips;
```

7.2.7 Check the load history.

```
SELECT *
   FROM TABLE(information_schema.copy_history(
     TABLE_NAME =>'trips',
     START_TIME => dateadd(hours, -1, current_timestamp()))
);
```

8 High-Performance Functions

This lab will take approximately *25 minutes* to complete.

The purpose of this exercise is to test Snowflake's high-performing functions and data sampling.

8.1 Explore the Sample Function

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


```
USE ROLE training_role;
USE WAREHOUSE [login]_QUERY_WH;
USE DATABASE snowflake_sample_data;
USE SCHEMA TPCH_SF10;
```

8.1.2 Select a sample from the lineitem table (each row has a 20% chance of being included):

```
SELECT * FROM lineitem SAMPLE (20);
```

By default, the SAMPLE function uses the row method - which means every row is examined to determine if it is in the sample set, or not. The row method is very accurate, but is also slower than the block method (which evaluates a block of rows at a time).

8.1.3 Select a sample from the lineitem table using the block method:

```
SELECT * FROM lineitem SAMPLE BLOCK (20);
```

Compare the speed of the row and block methods.

8.1.4 Sample with a seed option to produce deterministic behavior:

Run with the seed several times to verify that the samples are the same.

```
SELECT * FROM lineitem SAMPLE (10) SEED (4444);

SELECT * FROM lineitem SAMPLE (10) SEED (4444);

SELECT * FROM lineitem SAMPLE (10) SEED (4444);
```

8.1.5 JOIN samples from two different tables:

```
SELECT * FROM lineitem SAMPLE (5) JOIN orders SAMPLE (10)
      ON (l_orderkey = o_orderkey);
```

8.1.6 Use sampling to create training (90%) and test (remaining 10%) datasets for data science modeling:

```
CREATE TEMPORARY TABLE [login]_db.public.lineitem_training
  AS SELECT * FROM  lineitem SAMPLE (90);

CREATE TEMPORARY TABLE [login]_db.public.lineitem_testing
  AS SELECT * FROM lineitem
      WHERE l_orderkey NOT IN (SELECT l_orderkey
  FROM [login]_db.public.lineitem_training);
```


8.2 Use the Hyperloglog Approximate Count Function

Snowflake uses HyperLogLog to estimate the approximate number of distinct values in a data set. HyperLogLog is a state-of-the-art cardinality estimation algorithm, capable of estimating distinct cardinalities of trillions of rows with an average relative error of a few percent.

8.2.1 Resize your virtual warehouse for this test:

```
ALTER WAREHOUSE [login]_query_wh
  SET WAREHOUSE_SIZE = 'XLARGE';

ALTER SESSION SET USE_CACHED_RESULT = FALSE;
-- Disable reuse cached query results

ALTER WAREHOUSE [login]_query_wh SUSPEND;
-- Ignore error if the the warehouse is already in the suspended state

ALTER WAREHOUSE [login]_query_wh RESUME;
```

8.2.2 Determine an approximate count of distinct values with the HLL function:

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

8.2.3 View the query profile to see how long it took.

From looking at the query profile, the bulk of the execution time at first run is dominated by the table scan of the large data set, which obscures the speed of the HLL function. Note that 0% of the data was scanned from the data cache.

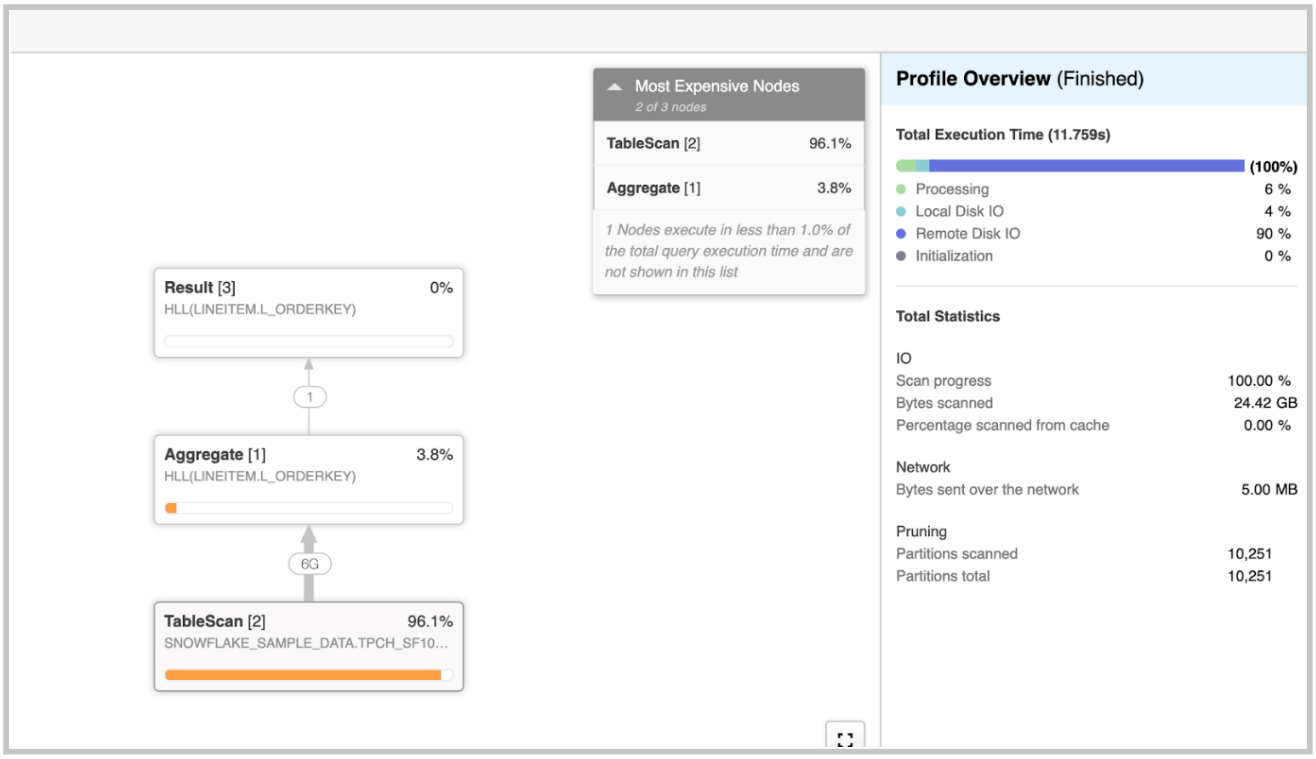

Figure 4: HLL Query Profile

8.2.4 Execute the same query again and record the execution time:

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

Since the the session has been altered not to reuse the query result cache, this query will not use the query result cache. However, this will use the data cache that was built up from the first run on the function. This eliminates the time required to scan the data, and gives a truer estimate of the speed of the HLL function.

Confirm that the query profile shows a very high Percentage scanned from cache (it should be over 98%).

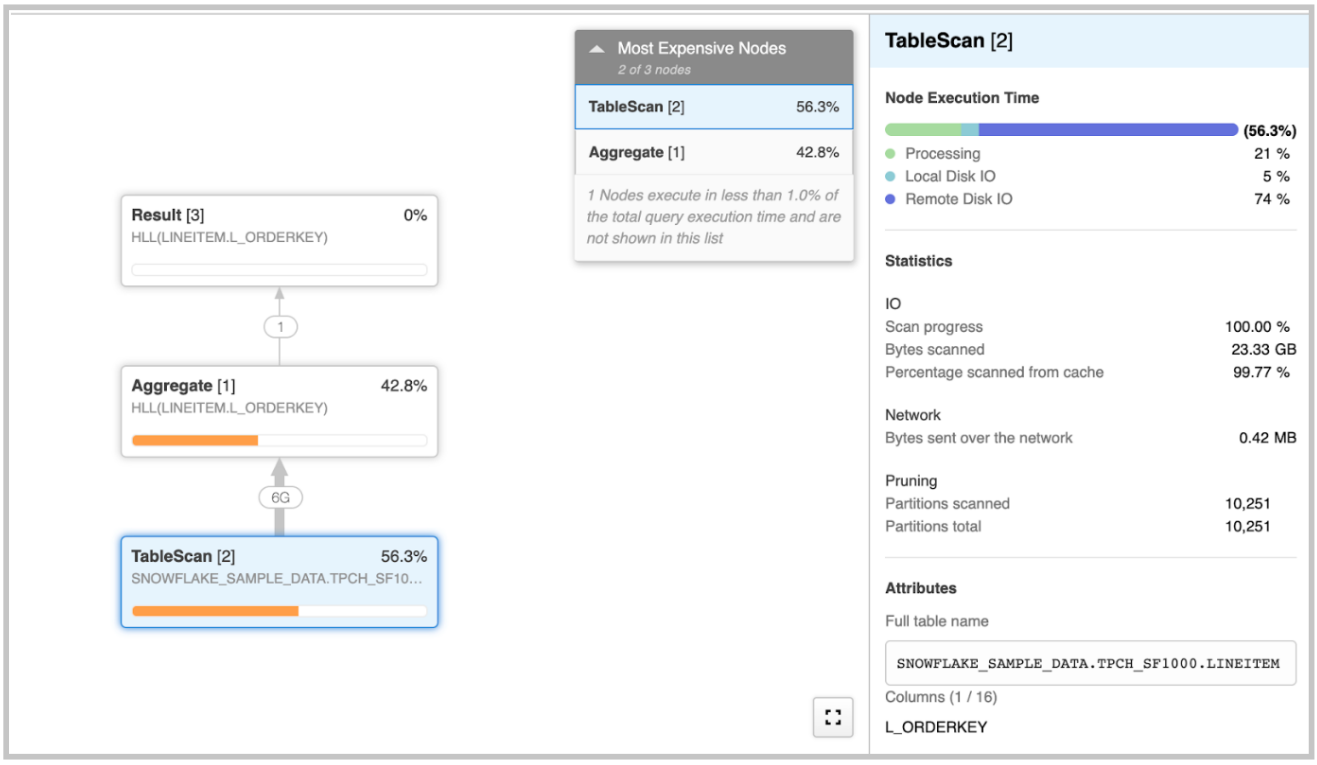

Figure 5: HLL Query Profile

8.2.5 Execute the regular count distinct version of the query:

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

8.2.6 Review the query profile to verify execution time, and cache used.

The query profile should again show that a high percentage of the data was scanned from cache. Since both functions used primarily data cache, it eliminates the time associated with the table scan.

Compare the execution time of the two queries and note the HLL approximate count version is much faster than the regular count version.

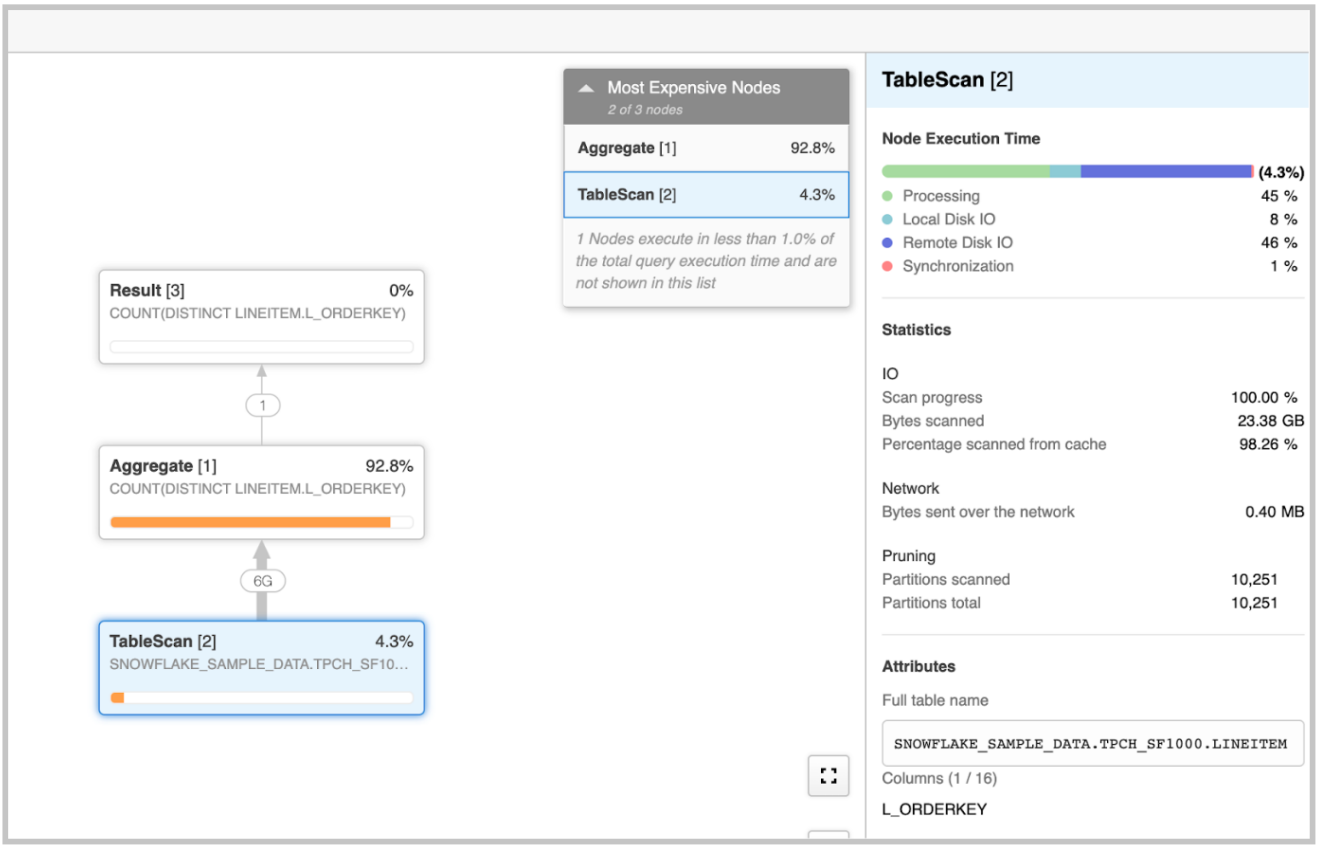

Figure 6: HLL Query Profile

8.3 Use the Percentile Estimation Function

8.3.1 Run a query using the SQL MEDIAN function:

If the select statement runs more than 5 minutes, cancel it.

```
USE SCHEMA snowflake_sample_data.tpcds_sf10tcl;

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

Review the time it took to complete as well as the figure.

8.3.2 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;
```


8.3.3 Review the time it took to complete.

Notice that the approximate percentile function was faster, and it returned a number almost identical to that of MEDIAN, thus making it more efficient.

8.4 Use Collations

Text strings in Snowflake are stored using the UTF-8 character set and, by default, strings are compared according to the Unicode codes that represent the characters in the string. Comparing strings based on their UTF-8 character representations might not provide the desired result.

In this task you will use collations to modify how string values are sorted and compared.

8.4.1 Set your context:

```
CREATE DATABASE IF NOT EXISTS [login]_db;
USE DATABASE [login]_db;
CREATE SCHEMA IF NOT EXISTS collation;
USE SCHEMA collation;
CREATE OR REPLACE WAREHOUSE [login]_QUERY_WH;
USE WAREHOUSE [login]_QUERY_WH;
ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE = 'XSMALL';
```

8.4.2 Create a source table named collation_tbl:

```
CREATE OR REPLACE TABLE collation_tbl(
    DEF_COLLATION STRING
  , CASE_INSENSITIVE STRING COLLATE 'EN-CI'
);
```

The first column uses the default collation (UTF-8), and the second column uses English, case-insensitive collation rules.

8.4.3 Insert some values into collation_tbl:

```
INSERT INTO collation_tbl(def_collation, case_insensitive)
  VALUES ('abc','AbC'),
         ('bcd','bcD'),
         ('ggg','GGG'),
         ('AAA','AAA'),
         ('aaa','aaa'),
         ('zzz','AAA'),
         ('ZZZ','zzz');
```


8.4.4 Select data without a sort:

```
SELECT * FROM collation_tbl;
```

The data is returned in the order it was ingested.

8.4.5 Select data sorted by the def_collation column:

```
SELECT * FROM collation_tbl ORDER BY def_collation;
```

The values in the def_collation column will have all strings that start with upper-case letters ordered first.

8.4.6 Select data sorted by case_insensitive column:

```
SELECT * FROM collation_tbl ORDER BY case_insensitive;
```

The values in the case_insensitive column are ordered without regard to the case of the first letter.

8.4.7 Change the sort order of the def_collation column using English case-insensitive collation:

```
SELECT * FROM collation_tbl ORDER BY COLLATE(def_collation,'en-ci');
```

8.4.8 Change the sort order of the def_collation column using English collation:

```
SELECT * FROM collation_tbl ORDER BY COLLATE(def_collation,'en');
```

Compare how the English collation is sorted in contrast to the English, case-insensitive collation.

8.5 Query Hierarchical Data

A common use case in SQL is creating a hierarchical structure out of a relational table. In the presentation you saw an example with employees, managers, and their reporting structure. This lab presents that example. Another use case is in banking where a bank wants to create a hierarchical structure with portfolio managers and their customer's investment portfolios.

8.5.1 Set your context:

```
USE ROLE training_role;
CREATE SCHEMA [login]_db.hierarchy;
USE SCHEMA [login]_db.hierarchy;
USE WAREHOUSE [login]_query_wh;
```


8.5.2 Create a table to hold employee data:

```
CREATE OR REPLACE TABLE employees (
      title STRING
    , employee_Id INTEGER
    , manager_Id INTEGER
);
```

8.5.3 Load data into the employees table:

```
INSERT INTO employees
VALUES ('CEO', 1, NULL)
      ,('CFO', 2, 1)
      ,('IT MANAGER', 3, 1)
      ,('IT ANALYST', 4, 3);

SELECT * FROM employees;
```

8.5.4 Query the hierarchical data using recursive WITH:

```
WITH hierarchy AS (
  SELECT
   *, 1 level, '/' || title chain
  FROM employees
  WHERE employee_id = 1
 UNION ALL
  SELECT
    e.*, h.level + 1,
    chain || '/' || e.title
  FROM
   employees e JOIN hierarchy h
   ON (e.manager_Id = h.employee_Id)
)

SELECT * FROM hierarchy;
```

8.5.5 Query the hierarchical data using “CONNECT BY”:

```
SELECT
  *, level, sys_connect_by_path(title, '/') chain
FROM employees
CONNECT BY prior employee_Id = manager_Id
START WITH employee_Id = 1;
```

9 Query Semi-Structured and Structured Data Together

This lab will take approximately *20 minutes* to complete.

Snowflake provides native support for semi-structured data. This module will walk you through the steps to:

- Explore the JSON-formatted historical weather data in the TRAINING_DB database.
- Query semi-structured data using SQL dot notation and the FLATTEN table function.
- Create a secure view that joins JSON data to the TRIPS table.

The JSON data consists of historical weather station observation data from numerous sources. The information is provided by NOAA Integrated Surface Database (ISD) on the Registry of Open Data on AWS NOAA Integrated Surface Database (ISD). The dataset includes over 35,000 stations worldwide, with some having data as far back as 1901, though the data show a substantial increase in volume in the 1940s and again in the early 1970s. This data set has been loaded in JSON format in the database named TRAINING_DB under the WEATHER schema.

9.1 Query Weather Data

9.1.1 Set your context.

```
USE ROLE TRAINING_ROLE;
USE DATABASE [login]_DB;
USE SCHEMA [login]_db.public;
USE WAREHOUSE [login]_QUERY_WH;
```

9.1.2 Query the contents of the TRAINING_DB.WEATHER.ISD_TOTAL table.

```
SELECT * FROM TRAINING_DB.WEATHER.ISD_TOTAL LIMIT 500;
```

9.1.3 Click on one of the records in the result set.

The data is stored in raw JSON format

Details


```
{
  "data": {
    "observations": [
      {
        "air": {
          "dew-point": 999.9,
          "dew-point-quality-code": "9",
          "temp": 11.1,
          "temp-quality-code": "1"
        },
        "atmospheric": {
          "pressure": 9976,
          "pressure-quality-code": "1"
        },
        "dt": "1911-09-04T20:00:00",
        "sky": {
          "ceiling": 99999,
          "ceiling-quality-code": "9"
        },
        "visibility": {
          "distance": 0,
          "distance-quality-code": "1"
        },
        "wind": {
          "direction-angle": 360,
          "direction-quality-code": "1",
          "speed-quality-code": "1",
          "speed-rate": 21
        }
      }
    ]
  },
  "station": {
    "USAF": "228920",
    "WBAN": 99999,
    "coord": {
      "lat": 60.717,
      "lon": 28.733
    },
    "country": "RS",
    "elev": 13,
    "id": "22892099999",
    "name": "VYBORG"
  }
}
```


Figure 7: Weather data Raw JSON

9.1.4 DESCRIBE the table.

```
DESCRIBE TABLE TRAINING_DB.WEATHER.ISD_TOTAL;
```

Note that JSON data is stored in the V column with the VARIANT data type.

9.2 Query semi-structured data

9.2.1 Query the JSON data in TRAINING_DB.WEATHER.ISD_TOTAL directly using SQL.

```
SELECT
       v:data.observations[0].dt::TIMESTAMP_NTZ AS observation_time,
       v:station.name::STRING AS station,
```



```
       v:station.country::STRING AS country,
       v:station.state::STRING AS state,
       v:station.id::STRING AS id,
       v:station.elev::FLOAT AS elevation,
       v:station.coord.lat::FLOAT AS lat,
       v:station.coord.lon::FLOAT AS lon,
       v:station.USAF::STRING AS USAF,
       v:station.WBAN::STRING AS WBAN,
       v:data.observations[0].air.temp::FLOAT AS temp_celsius,
       v:data.observations[0].air."temp-quality-code"::STRING AS temp_qc_code,
       v:data.observations[0].air."dew-point"::FLOAT AS dew_point,
       v:data.observations[0].air."dew-point-quality-code"::STRING AS
          dew_point_qc_code,
       v:data.observations[0].atmospheric.pressure::FLOAT AS atm_pressure,
       v:data.observations[0].atmospheric."pressure-quality-code"::STRING AS
          atm_pressure_qc_code,
       v:data.observations[0].sky.ceiling::FLOAT AS sky_ceiling,
       v:data.observations[0].sky."ceiling-quality-code"::STRING AS
          sky_ceiling_qc_code,
       v:data.observations[0].visibility.distance::FLOAT AS vis_distance,
       v:data.observations[0].visibility."distance-quality-code"::STRING AS
          vis_distance_qc_code,
       v:data.observations[0].wind."direction-angle"::FLOAT AS wind_direction,
       v:data.observations[0].wind."direction-quality-code"::STRING AS
          wind_direction_qc_code,
       v:data.observations[0].wind."speed-quality-code"::STRING AS
          wind_speed_qc_code,
       v:data.observations[0].wind."speed-rate"::FLOAT AS wind_speed
FROM TRAINING_DB.WEATHER.ISD_TOTAL
WHERE id = '74486094789'
LIMIT 500;
```

You can use SQL dot notation (e.g., station.coord.lat) to extract values from lower levels in the JSON document hierarchy. This allows Snowflake to treat each field as if it were a column in a relational table.

In fact, the results look similar to a regular structured data source:

Row	OBSERVATION_	STATION	COUNTRY	STATE	ID	ELEVATION	LAT	LON	USAF	WBAN	TEMP_CELSIUS	TEMP_QC_CODE	DEW_POINT	DEW_POINT_QC
1	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	20	5	15.6	5
2	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	20	1	15.6	1
3	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	21	5	15	5
4	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	20.6	5	15	5
5	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22.8	5	15	5
6	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22	5	15	5
7	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22.2	5	15	5
8	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22.2	1	15	1
9	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22.2	5	15	5
10	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	23.3	5	15	5
11	2014-08-23 ...	JOHN F KEN...	US	NY	74486094789	3.4	40.639	-73.764	744860	94789	22.8	5	16.1	5

Figure 8: JSON Query Results

9.2.2 Query a portion of the data to see the name, country, and ID for stations, as well as the observations array.

```
SELECT
     v:station.name::STRING AS station,
     v:station.country::STRING AS country,
     v:station.id::STRING AS id,
     v:data.observations AS observations
FROM TRAINING_DB.WEATHER.ISD_DAILY
LIMIT 500;
```

OBSERVATIONS is an array with multiple JSON objects.

Row	STATION	COUNTRY	ID	OBSERVATIONS
1	NADI INTL	FJ	91680099999	[{ "air": { "dew-point": 21.1, "dew-point-quality-code": "1", "temp": 31.7...
2	SAMSUN	TU	17030099999	[{ "air": { "dew-point": 1.1, "dew-point-quality-code": "1", "temp": 18.9, ...
3	IM E K FEDOROVA	RS	20946099999	[{ "air": { "dew-point": -22.2, "dew-point-quality-code": "1", "temp": -2...
4	UST'-JUDOMA	RS	31054099999	[{ "air": { "dew-point": -40, "dew-point-quality-code": "1", "temp": -37....
5	SAN NICHOLAS ISLAND	US	99999993116	[{ "air": { "dew-point": 7.8, "dew-point-quality-code": "5", "temp": 12.8,...
6	PRESIDENTE PERON	AR	87715099999	[{ "air": { "dew-point": 14.4, "dew-point-quality-code": "1", "temp": 22....
7	HANAU AAF	GM	10642099999	[{ "air": { "dew-point": 6.3, "dew-point-quality-code": "1", "temp": 7.4, ...
8	CEDARA	SF	68580099999	[{ "air": { "dew-point": 15.6, "dew-point-quality-code": "1", "temp": 23....

Figure 9: JSON Array Query Results

9.2.3 Use the FLATTEN table function to flatten (explodes) the OBSERVATIONS array into multiple rows.

```
SELECT weather.t as date,
       v:station.name::STRING AS station,
       v:station.country::STRING AS country,
       v:station.id::STRING AS id,
       observations.value:air.temp::FLOAT AS temp_celsius,
       observations.value:air."temp-quality-code"::STRING AS temp_qc_code,
       observations.value:air."dew-point"::FLOAT AS dew_point,
       observations.value:air."dew-point-quality-code"::STRING AS
          dew_point_qc_code,
       observations.value:atmospheric.pressure::FLOAT AS atm_pressure,
       observations.value:atmospheric."pressure-quality-code"::STRING AS
          atm_pressure_qc_code,
       observations.value:sky.ceiling::FLOAT AS sky_ceiling,
       observations.value:sky."ceiling-quality-code"::STRING AS
          sky_ceiling_qc_code,
       observations.value:visibility.distance::FLOAT AS vis_distance,
       observations.value:visibility."distance-quality-code"::STRING AS
          vis_distance_qc_code,
       observations.value:wind."direction-angle"::FLOAT AS wind_direction,
       observations.value:wind."direction-quality-code"::STRING AS
          wind_direction_qc_code,
       observations.value:wind."speed-quality-code"::STRING AS wind_speed_qc_code,
       observations.value:wind."speed-rate"::FLOAT AS wind_speed
FROM TRAINING_DB.WEATHER.isd_daily weather,
```



```
LATERAL FLATTEN(input => v:data.observations) observations
LIMIT 500;
```

The command above uses the FLATTEN function to extract values into multiple rows. This allows Snowflake to treat each field as if it were a column in a relational table. The command also uses the :: operator to cast values to different data types.

The results look similar to a regular structured data source:

Row	DATE	STATION	COUNTRY	ID	TEMP_CELSIUS	TEMP_QC_CODE	DEW_POINT	DEW_POINT_QC	ATM_PRESSURE	ATM_PRESSURE	SKY_CEILING	SKY_
1	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	6.7	5	5.6	5	10072	5	518	4
2	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	7.8	5	7.2	5	10044	5	1219	4
3	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	3.9	5	1.1	5	10091	5	762	4
4	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	1.7	5	-3.3	5	10132	5	1067	4
5	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	0.6	5	-6.1	5	10167	5	914	4
6	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	0	5	-7.2	5	10194	5	914	4
7	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	1.1	5	-7.2	5	10201	5	1067	4
8	1965-11-09	BURLNGTN ETAN ALN AP	US	99999914742	0	5	-7.8	5	10221	5	1219	4
9	1965-11-09	KERMAN	IR	40841099999	17.8	1	999.9	9	10122	1	9000	1
10	1965-11-09	KERMAN	IR	40841099999	22.2	1	-11.1	1	10086	1	3600	1
11	1965-11-09	CAPE PESCHANIY	RS	20097099999	-13.9	1	-15	1	10241	1	2400	1

Figure 10: Flatten JSON Array Query Results

9.3 Create a View for Production

9.3.1 Create a view containing weather data from the station at JFK airport in New York (station ID 74486094789).

This statement defines a view named WEATHER_NY_VW. This view calculates the maximum and minimum temperatures observed at JFK Airport in New York after 2015-01-01, and converts the air temperature from celsius to fahrenheit.

```
CREATE OR REPLACE VIEW WEATHER_NY_DEV AS
SELECT weather.t as date,
       (MAX(observations.value:air.temp) * 9/5 + 32)::NUMBER(38,1) as max_temp_f,
       (MIN(observations.value:air.temp) * 9/5 + 32)::NUMBER(38,1) as min_temp_f
FROM TRAINING_DB.WEATHER.isd_daily weather,
LATERAL FLATTEN(input => v:data.observations) observations
WHERE observations.value:air."temp-quality-code" = '1'
    AND date >= to_date('2015-01-01')
    AND weather.v:station.id = '74486094789'
GROUP BY 1;
```

9.3.2 Query the view to verify that it provides the desired data.

```
SELECT * FROM WEATHER_NY_DEV
WHERE DATE_TRUNC ('month', date) = '2018-01-01'
LIMIT 50;
```


9.4 Create a Joined View

9.4.1 Run the following statement to create a view joining the WEATHER_VW and TRIPS tables:

```
CREATE OR REPLACE VIEW TRIP_WEATHER_VW
    AS SELECT * FROM CITIBIKE.SCHEMA1.TRIPS T
LEFT OUTER JOIN WEATHER_NY_DEV W
   ON TO_DATE(T.STARTTIME) = TO_DATE(W.DATE);
```

9.4.2 Use the view to count the total number of trips per day and the maximum and minimum temperatures on that day.

```
SELECT
    COUNT(*) AS NUM_TRIPS,
    DATE,
    MAX_TEMP_F,
    MIN_TEMP_F
FROM TRIP_WEATHER_VW
WHERE DATE IS NOT NULL
GROUP BY 2, 3, 4
ORDER BY 1 DESC;
```

10 Work with External Tables

This lab will take approximately *35 minutes* to complete.

10.1 Unload Data to Cloud Storage as a Data Lake

To start, you will unload data from the Citibike table onto an external stage.

10.1.1 Set your context.

```
--Note that we can select the database and schema in one statement
USE ROLE training_role;
CREATE SCHEMA IF NOT EXISTS [login]_db.CITIBIKE;
USE schema [login]_db.CITIBIKE;
CREATE WAREHOUSE IF NOT EXISTS [login]_QUERY_WH;
USE WAREHOUSE [login]_QUERY_WH;
```

10.1.2 Determine the size of the Citibike database.

Create a query to get row count and size from Citibike table. Record the results for later.


```
SELECT TABLE_NAME
     , ROW_COUNT
     , BYTES
     , BYTES / (50*1024*1024) as NUM_CHUNKS
     , NUM_CHUNKS/8 as MAX_NODES
   FROM SNOWFLAKE.ACCOUNT_USAGE.TABLES
   WHERE table_name like 'TRIPS'
     AND table_schema like 'SCHEMA1'
     AND table_catalog like 'CITIBIKE';
```

The result likely indicates that the maximum number of nodes required is between 4 and 5. A medium cluster is 4 nodes, and a large cluster is 8 nodes. A large cluster should run the dump faster, but the medium will use fewer credits.

10.2 Unload Data to Cloud Storage with Different Warehouse Sizes

10.2.1 Set your context.

```
USE ROLE training_role;
USE SCHEMA citibike.schema1;
USE WAREHOUSE [login]_load_wh;
```

10.2.2 Set the warehouse size to MEDIUM for the first test.

```
ALTER WAREHOUSE [login]_load_wh SET WAREHOUSE_SIZE=MEDIUM;
```

10.2.3 Copy data to the external stage.

```
COPY INTO
    '@TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike1'
  FROM (SELECT * FROM CITIBIKE.SCHEMA1.TRIPS)
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYGZIPPIPEFORMAT)
  MAX_FILE_SIZE=49000000
;

ls @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike1;
```

10.2.4 Set the warehouse size to LARGE for the second test.

```
ALTER WAREHOUSE [login]_load_wh SUSPEND;

ALTER WAREHOUSE [login]_load_wh SET WAREHOUSE_SIZE=LARGE;
```


10.2.5 Copy the data to the stage.

```
COPY INTO
    '@TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike2'
  FROM (SELECT * FROM CITIBIKE.SCHEMA1.TRIPS)
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYGZIPPIPEFORMAT)
  MAX_FILE_SIZE=49000000
;

ls @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike2;
```

10.2.6 View the query profiles to see the performance difference.

Looking at the results of the two approaches we do see that the large cluster took less clock time than the medium cluster. We also see files of varying sizes. Both approaches ended up with approximately 64 files.

10.2.7 Clean up the stage.

```
remove
   @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike1;

remove
   @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike2;
```

10.2.8 Unload the CITIBIKE data into Parquet files.

```
COPY INTO
    '@TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike/trips'
  FROM (SELECT * FROM CITIBIKE.SCHEMA1.TRIPS)
  FILE_FORMAT=(FORMAT_NAME=training_db.traininglab.MYPARQUETFORMAT)
  MAX_FILE_SIZE=49000000;

ls @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike;
```

10.2.9 Create an external table using the unloaded Parquet files.

```
USE DATABASE [login]_DB;
CREATE OR REPLACE EXTERNAL TABLE EXT_PARQUET_TRIPS
  location=
     @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/citibike/
  FILE_FORMAT=(TYPE=PARQUET);

ALTER EXTERNAL TABLE EXT_PARQUET_TRIPS REFRESH;

SELECT * FROM EXT_PARQUET_TRIPS LIMIT 10;
```


10.2.10 Unload the data using a join between the tpch customer and nation tables.

```
COPY INTO
    '@TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/parquet/tpch/myjoin'
  FROM (SELECT c_name
     , c_nationkey
     , c_address
     , c_acctbal
     , n_nationkey
     , n_name
  FROM snowflake_sample_data.tpch_sf1.customer join
     snowflake_sample_data.tpch_sf1.nation
    on c_nationkey = n_nationkey)
  FILE_FORMAT=(TYPE='PARQUET');

CREATE OR REPLACE EXTERNAL TABLE EXT_PARQUET_myjoin
  location=
     @TRAINING_DB.TRAININGLAB.CLASS_STAGE/COURSE/ADVANCED/student/[login]/parquet/tpch/
  FILE_FORMAT=(TYPE=PARQUET);

ALTER EXTERNAL TABLE EXT_PARQUET_myjoin REFRESH;

SELECT * FROM EXT_PARQUET_myjoin limit 10;
```

10.3 Execute Queries Against External Tables and Metadata

10.3.1 Set your context.

```
USE ROLE training_role;
USE WAREHOUSE [login]_query_wh;
CREATE DATABASE [login]_tpcdi_stg;
USE SCHEMA public;
```

10.3.2 List the staged files.

```
LIST @training_db.traininglab.ed_stage/finwire;
```

10.3.3 Create a file format to query the data.

```
CREATE OR REPLACE FILE FORMAT [login]_TPCDI_STG.PUBLIC.TXT_FIXED_WIDTH
  TYPE = CSV
  COMPRESSION = 'AUTO'
  FIELD_DELIMITER = NONE
  RECORD_DELIMITER = '\\n'
  SKIP_HEADER = 0
  TRIM_SPACE = FALSE
  ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE
  NULL_IF = ('\\N');
```


10.3.4 Create an external table.

```
CREATE OR REPLACE EXTERNAL TABLE finwire
  LOCATION = @training_db.traininglab.ed_stage/finwire
  REFRESH_ON_CREATE = FALSE
  FILE_FORMAT = (FORMAT_NAME = 'txt_fixed_width');
```

10.3.5 Explore TABLE and EXTERNAL TABLE metadata.

```
SHOW TABLES;
SHOW EXTERNAL TABLES;
```

10.3.6 Execute a simple query against the external table.

```
SELECT * FROM finwire LIMIT 10;
```

Note: There are no results because `REFRESH_ON_CREATE = FALSE` was specified when the table was created.

10.3.7 Manually refresh the external table metadata.

```
ALTER EXTERNAL TABLE finwire REFRESH;
```

Refreshing the external table synchronizes the metadata with the current list of data files in the specified stage. This action is required for the metadata to register any existing data files in the named external stage.

You should see output similar to the following, showing the tables loaded to the external table.

Query ID SQL 1.42s 203 rows

Filter result... Copy Col

Row	file	status	description
1	modules/finwire/FINWIRE1997Q1	REGISTERED_NEW	File registered successfully.
2	modules/finwire/FINWIRE1974Q2	REGISTERED_NEW	File registered successfully.
3	modules/finwire/FINWIRE1974Q1	REGISTERED_NEW	File registered successfully.
4	modules/finwire/FINWIRE2013Q1	REGISTERED_NEW	File registered successfully.
5	modules/finwire/FINWIRE2013Q3	REGISTERED_NEW	File registered successfully.
6	modules/finwire/FINWIRE2013Q2	REGISTERED_NEW	File registered successfully.
7	modules/finwire/FINWIRE1974Q4	REGISTERED_NEW	File registered successfully.
8	modules/finwire/FINWIRE1997Q4	REGISTERED_NEW	File registered successfully.
9	modules/finwire/FINWIRE1974Q3	REGISTERED_NEW	File registered successfully.
10	modules/finwire/FINWIRE2013Q4	REGISTERED_NEW	File registered successfully.

Figure 11: Query Results

10.3.8 Rerun the query.

```
SELECT * FROM finwire LIMIT 10;
```

The result set is a single variant column. Take a look at the query profile.

10.4 Work with External Tables

10.4.1 Create a table with columns.

```
create or replace external table finwire
    (
      PTS VARCHAR(15) AS SUBSTR($1, 8, 15)
    , REC_TYPE VARCHAR(3) AS SUBSTR($1, 23, 3)
    , COMPANY_NAME VARCHAR(60) AS SUBSTR($1, 26, 60)
    , CIK VARCHAR(10) AS SUBSTR($1, 86, 10)
    , STATUS VARCHAR(4) AS IFF(SUBSTR($1, 23, 3) = 'CMP', SUBSTR($1, 96,
       4),SUBSTR($1, 47, 4))
    , INDUSTRY_ID VARCHAR(2) AS SUBSTR($1, 100, 2)
    , SP_RATING VARCHAR(4) AS SUBSTR($1, 102, 4)
    , FOUNDING_DATE VARCHAR(8) AS SUBSTR($1, 106, 8)
    , ADDR_LINE1 VARCHAR(80) AS SUBSTR($1, 114, 80)
    , ADDR_LINE2 VARCHAR(80) AS SUBSTR($1, 194, 80)
    , POSTAL_CODE VARCHAR(12) AS SUBSTR($1, 274, 12)
    , CITY VARCHAR(25) AS SUBSTR($1, 286, 25)
    , STATE_PROVINCE VARCHAR(20) AS SUBSTR($1, 311, 20)
    , COUNTRY VARCHAR(24) AS SUBSTR($1, 331, 24)
    , CEO_NAME VARCHAR(46) AS SUBSTR($1, 355, 46)
    , DESCRIPTION VARCHAR(150) AS SUBSTR($1, 401, 150)
    , YEAR VARCHAR(4) AS SUBSTR($1, 8, 4)
    , QUARTER VARCHAR(1) AS SUBSTR($1, 30, 1)
    , QTR_START_DATE VARCHAR(8) AS SUBSTR($1, 31, 8)
    , POSTING_DATE VARCHAR(8) AS SUBSTR($1, 39, 8)
    , REVENUE VARCHAR(17) AS SUBSTR($1, 47, 17)
    , EARNINGS VARCHAR(17) AS SUBSTR($1, 64, 17)
    , EPS VARCHAR(12) AS SUBSTR($1, 81, 12)
    , DILUTED_EPS VARCHAR(12) AS SUBSTR($1, 93, 12)
    , MARGIN VARCHAR(12) AS SUBSTR($1, 105, 12)
    , INVENTORY VARCHAR(17) AS SUBSTR($1, 117, 17)
    , ASSETS VARCHAR(17) AS SUBSTR($1, 134, 17)
    , LIABILITIES VARCHAR(17) AS SUBSTR($1, 151, 17)
    , SH_OUT VARCHAR(13)AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 168, 13),
       SUBSTR($1, 127, 13))
    , DILUTED_SH_OUT VARCHAR(13) AS SUBSTR($1, 181, 13)
    , CO_NAME_OR_CIK VARCHAR(60) AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 194,
       10), SUBSTR($1, 168, 10))
    , SYMBOL VARCHAR(15) AS SUBSTR($1, 26, 15)
    , ISSUE_TYPE VARCHAR(6) AS SUBSTR($1, 41, 6)
    , NAME VARCHAR(70) AS SUBSTR($1, 51, 70)
    , EX_ID VARCHAR(6) AS SUBSTR($1, 121, 6)
    , FIRST_TRADE_DATE VARCHAR(8) AS SUBSTR($1, 140, 8)
    , FIRST_TRADE_EXCHG VARCHAR(8) AS SUBSTR($1, 148, 8)
    , DIVIDEND VARCHAR(12) AS SUBSTR($1, 156, 12)
```



```
    )
location = @training_db.traininglab.ed_stage/finwire
file_format = (format_name = 'txt_fixed_width');
```

10.4.2 Refresh the external table.

```
ALTER EXTERNAL TABLE finwire REFRESH;
```

10.4.3 Query the revised table.

```
SELECT *
   FROM finwire
   WHERE rec_type = 'CMP' limit 10;
SELECT co_name_or_cik
     , year
     , quarter
     , sum(revenue::number)
  FROM finwire
  WHERE rec_type='FIN' and year='1967' and quarter='2' group by 1,2,3;
```

Again, examine the query profile for each query and see how efficient the queries are in taking advantage of partition pruning.

10.4.4 Examine the file metadata.

```
SELECT year
     , quarter
     , co_name_or_cik
     , metadata$filename
  FROM finwire
  WHERE rec_type='FIN' LIMIT 10;
```

10.5 Create an External Table with Partitions Based on the File Name.

10.5.1 Create the external table.

```
create or replace external table finwire
(
    YEAR  VARCHAR(4) AS SUBSTR(METADATA$FILENAME, 16, 4)
  , QUARTER VARCHAR(1) AS SUBSTR(METADATA$FILENAME, 21, 1)
  , thestring varchar(90) AS  SUBSTR(METADATA$FILENAME, 1, 50)
  , PTS VARCHAR(15) AS SUBSTR($1, 8, 15)
  , REC_TYPE VARCHAR(3) AS SUBSTR($1, 23, 3)
  , COMPANY_NAME VARCHAR(60) AS SUBSTR($1, 26, 60)
  , CIK VARCHAR(10) AS SUBSTR($1, 86, 10)
  , STATUS VARCHAR(4) AS IFF(SUBSTR($1, 23, 3) = 'CMP', SUBSTR($1, 96,
    4),SUBSTR($1, 47, 4))
```



```
  , INDUSTRY_ID VARCHAR(2) AS SUBSTR($1, 100, 2)
  , SP_RATING VARCHAR(4) AS SUBSTR($1, 102, 4)
  , FOUNDING_DATE VARCHAR(8) AS SUBSTR($1, 106, 8)
  , ADDR_LINE1 VARCHAR(80) AS SUBSTR($1, 114, 80)
  , ADDR_LINE2 VARCHAR(80) AS SUBSTR($1, 194, 80)
  , POSTAL_CODE VARCHAR(12) AS SUBSTR($1, 274, 12)
  , CITY VARCHAR(25) AS SUBSTR($1, 286, 25)
  , STATE_PROVINCE VARCHAR(20) AS SUBSTR($1, 311, 20)
  , COUNTRY VARCHAR(24) AS SUBSTR($1, 331, 24)
  , CEO_NAME VARCHAR(46) AS SUBSTR($1, 355, 46)
  , DESCRIPTION VARCHAR(150) AS SUBSTR($1, 401, 150)
  , QTR_START_DATE VARCHAR(8) AS SUBSTR($1, 31, 8)
  , POSTING_DATE VARCHAR(8) AS SUBSTR($1, 39, 8)
  , REVENUE VARCHAR(17) AS SUBSTR($1, 47, 17)
  , EARNINGS VARCHAR(17) AS SUBSTR($1, 64, 17)
  , EPS VARCHAR(12) AS SUBSTR($1, 81, 12)
  , DILUTED_EPS VARCHAR(12) AS SUBSTR($1, 93, 12)
  , MARGIN VARCHAR(12) AS SUBSTR($1, 105, 12)
  , INVENTORY VARCHAR(17) AS SUBSTR($1, 117, 17)
  , ASSETS VARCHAR(17) AS SUBSTR($1, 134, 17)
  , LIABILITIES VARCHAR(17) AS SUBSTR($1, 151, 17)
  , SH_OUT VARCHAR(13)AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 168, 13),
     SUBSTR($1, 127, 13))
  , DILUTED_SH_OUT VARCHAR(13) AS SUBSTR($1, 181, 13)
  , CO_NAME_OR_CIK VARCHAR(60) AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 194,
     10), SUBSTR($1, 168, 10))
  , SYMBOL VARCHAR(15) AS SUBSTR($1, 26, 15)
  , ISSUE_TYPE VARCHAR(6) AS SUBSTR($1, 41, 6)
  , NAME VARCHAR(70) AS SUBSTR($1, 51, 70)
  , EX_ID VARCHAR(6) AS SUBSTR($1, 121, 6)
  , FIRST_TRADE_DATE VARCHAR(8) AS SUBSTR($1, 140, 8)
  , FIRST_TRADE_EXCHG VARCHAR(8) AS SUBSTR($1, 148, 8)
  , DIVIDEND VARCHAR(12) AS SUBSTR($1, 156, 12)
)
partition by (year,quarter)
location = @training_db.traininglab.ed_stage/finwire
file_format = (format_name = 'txt_fixed_width');
```

10.5.2 Refresh the table.

```
ALTER EXTERNAL TABLE finwire REFRESH;
```

10.5.3 Execute some queries and examine their profiles

```
select co_name_or_cik
     , year
     , quarter
     , sum(revenue::number)
  from finwire
  where rec_type='FIN' and year='1967' and quarter='3' group by 1,2,3;

select co_name_or_cik
```



```
    , year
    , quarter
    , sum(revenue::number)
  from finwire
  where rec_type='FIN' and year='1989' and quarter='3' group by 1,2,3;
```

11 Query and Search Optimization

This lab will take approximately *40 minutes* to complete.

11.1 Explore Query Performance

11.1.1 Set your context and disable the query result cache.

```
USE role training_role;
USE warehouse [login]_QUERY_WH;
USE database training_db;
USE schema TPCH_SF1000;

ALTER SESSION SET USE_CACHED_RESULT = false;
```

11.1.2 Run a query that filters on columns that are well-clustered:

```
SELECT
  c_custkey,
  c_name,
  sum(l_extendedprice * (1 - l_discount)) as revenue, c_acctbal,
  n_name,
  c_address,
  c_phone,
  c_comment
FROM customer
  inner join orders
    on c_custkey = o_custkey
  inner join lineitem
    on l_orderkey = o_orderkey
  inner join nation
    on c_nationkey = n_nationkey
WHERE
  o_orderdate >= to_date('1993-10-01')
    AND o_orderdate < dateadd(month, 3, to_date('1993-10-01'))
    AND l_returnflag = 'R'
GROUP BY
  c_custkey,
  c_name,
  c_acctbal,
  c_phone,
  n_name,
  c_address,
```



```
  c_comment
ORDER BY
  3 desc
LIMIT 20;
```

11.1.3 View the query profile and note performance metrics.

Take note of:

- Partitions scanned
- Partitions total

How effective was micro-partition pruning for this query? This query was filtered on the O_ORDERDATE column. In general, date columns tend to be fairly well clustered.

11.1.4 Check the clustering quality of the o_orderdate column.

```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'orders' , '(o_orderdate)' );
```

11.1.5 Click on the result row and examine statistics.

Review the result and notice that the table is fairly well clustered around the o_orderdate dimension.

- The average_depth is relatively low, which indicates effective clustering
- The histogram shows most micro-partitions clustered at the top end (depths of 1 and 2), which also indicates effective clustering

```
/*
{
  "cluster_by_keys" : "LINEAR(o_orderdate)",
  "total_partition_count" : 3180,
  "total_constant_partition_count" : 1030,
  "average_overlaps" : 1.195,
  "average_depth" : 1.678,
  "partition_depth_histogram" : {
    "00000" : 0,
    "00001" : 1024,
    "00002" : 2156,
    "00003" : 0,
    "00004" : 0,
    "00005" : 0,
    "00006" : 0,
    "00007" : 0,
    "00008" : 0,
    "00009" : 0,
    "00010" : 0,
    "00011" : 0,
    "00012" : 0,
```



```
    "00013" : 0,
    "00014" : 0,
    "00015" : 0,
    "00016" : 0
  }
}
*/
```

11.1.6 Return to the query profile for the query you ran.

11.1.7 Click on the TableScan [6] operator (at the bottom of the query profile).

How many micro-partitions were pruned? Is this table scan filtered on a column that is well-clustered?

The SQL pruner did not skip any micro-partitions when reading the table CUSTOMER because there was no WHERE clause, and the JOIN condition was on a column that was not well clustered.

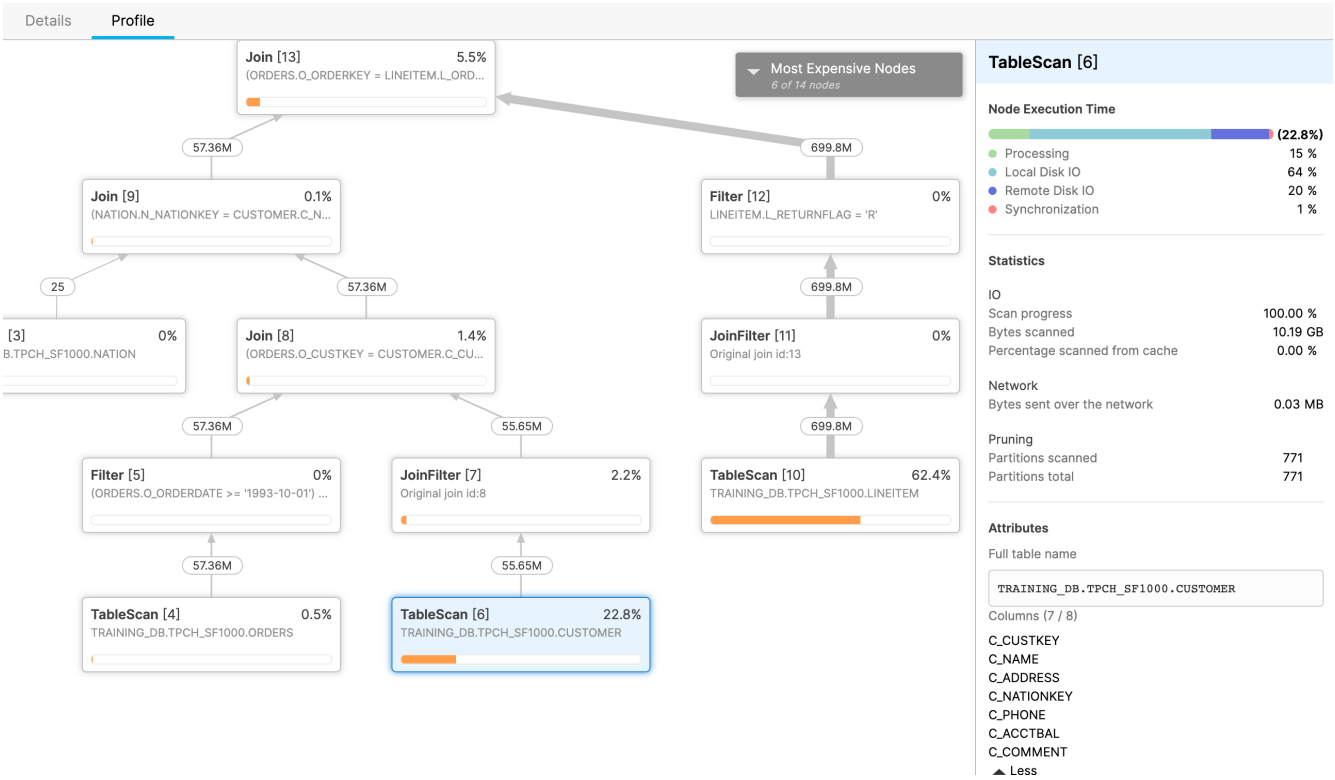

Figure 12: Table Scan Node

11.1.8 Check the clustering quality of the c_custkey column.

```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'customer' , '(c_custkey)' );
```


11.1.9 Open the row and examine the result.

- The clustering depth is high, which indicates a poorly-clustered column
- The histogram shows most of the micro-partitions grouped toward the bottom of the histogram, which is another indication of a poorly-clustered column.

11.2 Explore GROUP BY and ORDER BY Operation Performance

11.2.1 Set the warehouse size.

```
ALTER warehouse [login]_QUERY_WH SET WAREHOUSE_SIZE=SMALL;
```

11.2.2 Run a query which has a GROUP BY on a column with few distinct values.

```
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;
```

Note that this produces only four “groups.”

11.2.3 View the query profile and click on the operator **Aggregate [1]**.

Note that the amount of data shuffled during the parallel aggregation operation (`Bytes sent over the network`) is minimal.

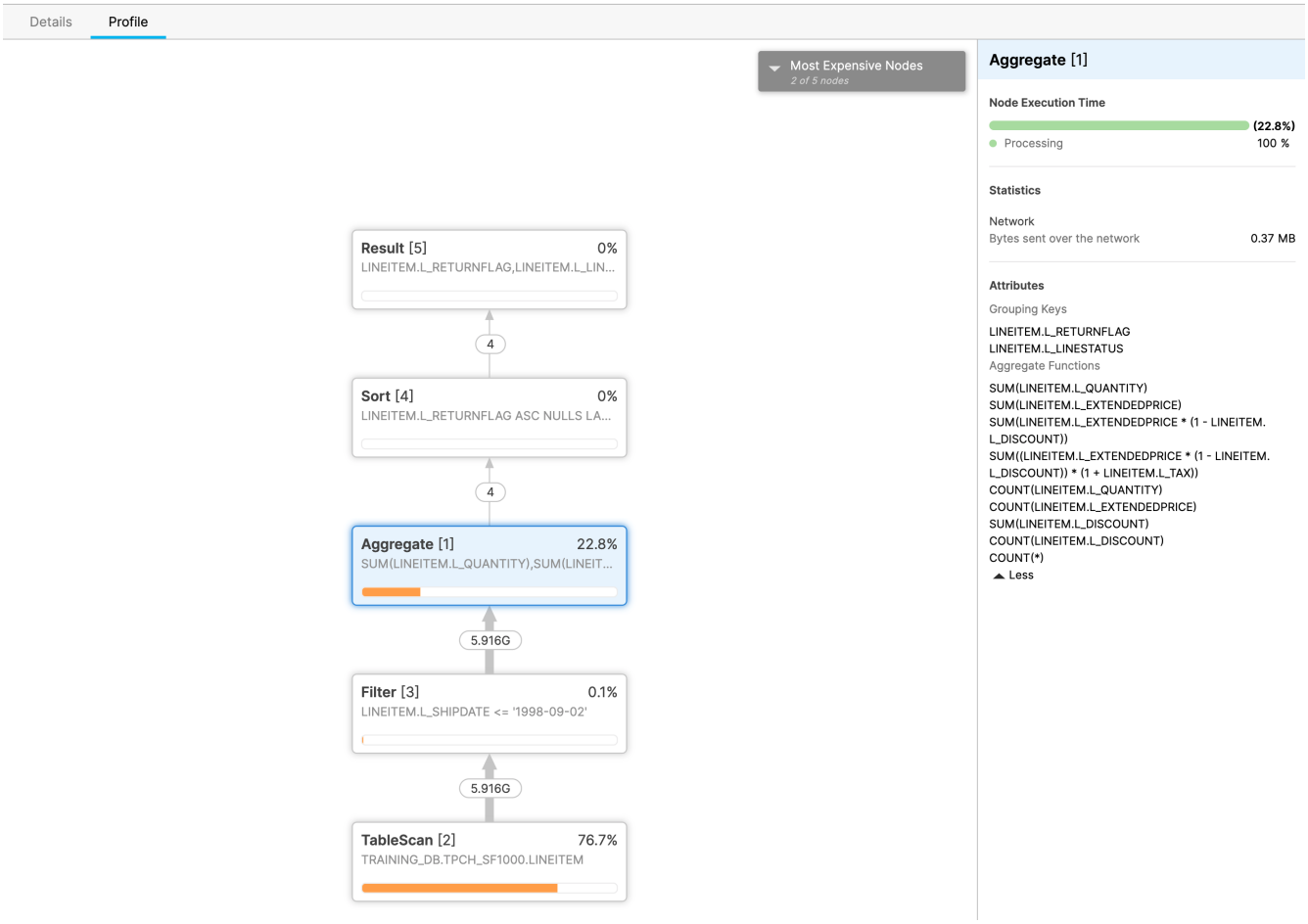

Figure 13: Aggregate Node

11.2.4 Click ouside the operator to show the statistics panel.

Note that there is no spilling to local or remote storage.

11.2.5 Click on the operator **Sort [4]**.

This is the operator for the ORDER BY. Note that the amount of data shuffled during the global sort operation is also minimal.

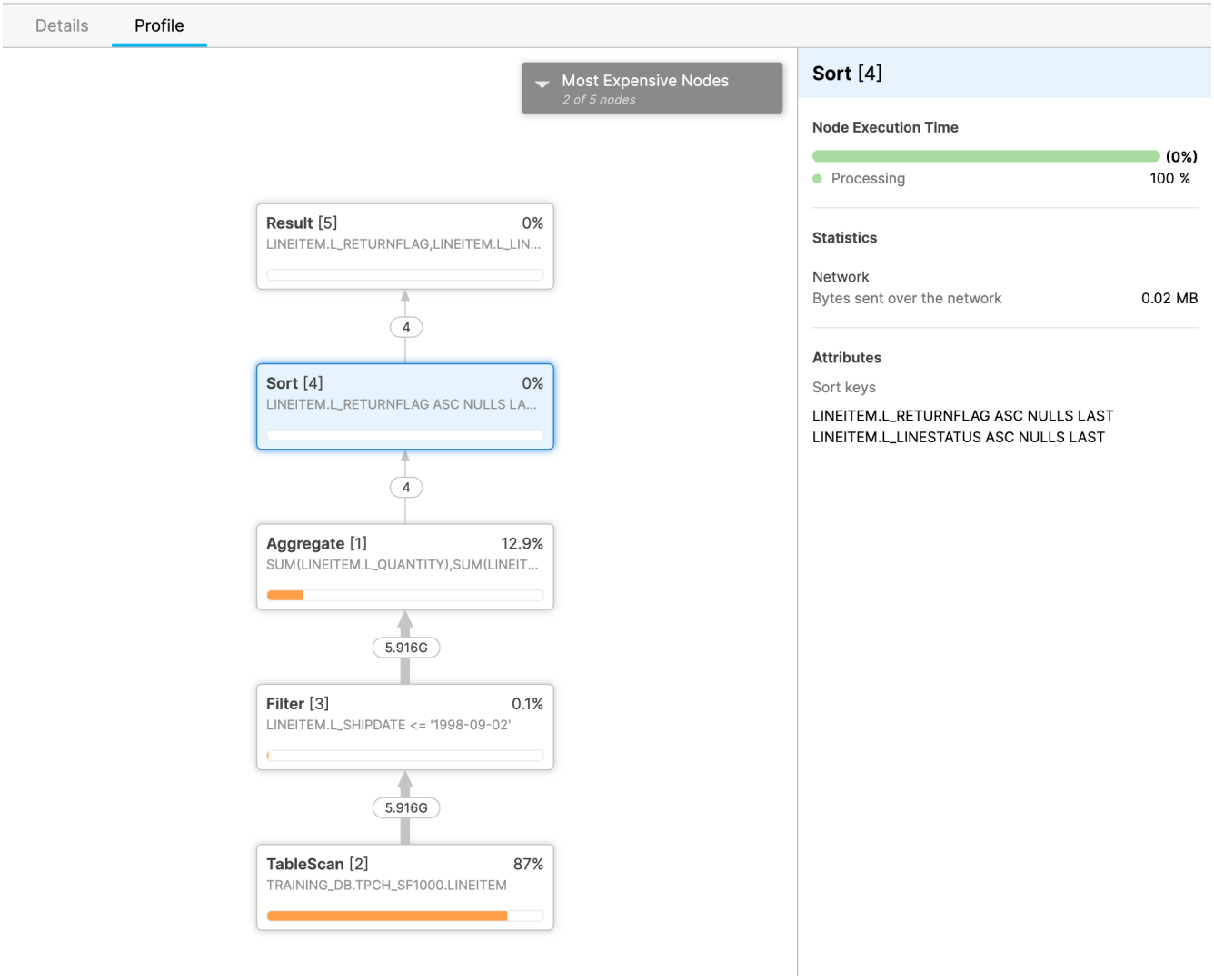

Figure 14: Sort Node

11.2.6 Run a query with a GROUP BY on a column with many distinct values.

```
SELECT l_shipdate, count( * ) FROM lineitem
GROUP BY 1
ORDER BY 1;
```

Note that this creates over 2,000 groups.

11.3 Querying with LIMIT

Applying a LIMIT clause to a query does not affect the amount of data that is read; it merely limits the results set output.

11.3.1 Execute the following query with a LIMIT clause.


```
SELECT
S.SS_SOLD_DATE_SK,
R.SR_RETURNED_DATE_SK,
S.SS_STORE_SK,
S.SS_ITEM_SK,
S.SS_CUSTOMER_SK,
S.SS_TICKET_NUMBER,
S.SS_QUANTITY,
S.SS_SALES_PRICE,
S.SS_CUSTOMER_SK,
S.SS_STORE_SK,
S.SS_QUANTITY,
S.SS_SALES_PRICE,
R.SR_RETURN_AMT
FROM SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE_SALES  S
INNER JOIN SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE_RETURNS  R on
   R.SR_ITEM_SK=S.SS_ITEM_SK
WHERE  S.SS_ITEM_SK =4164
LIMIT 100;
```


11.3.2 Review the query profile:

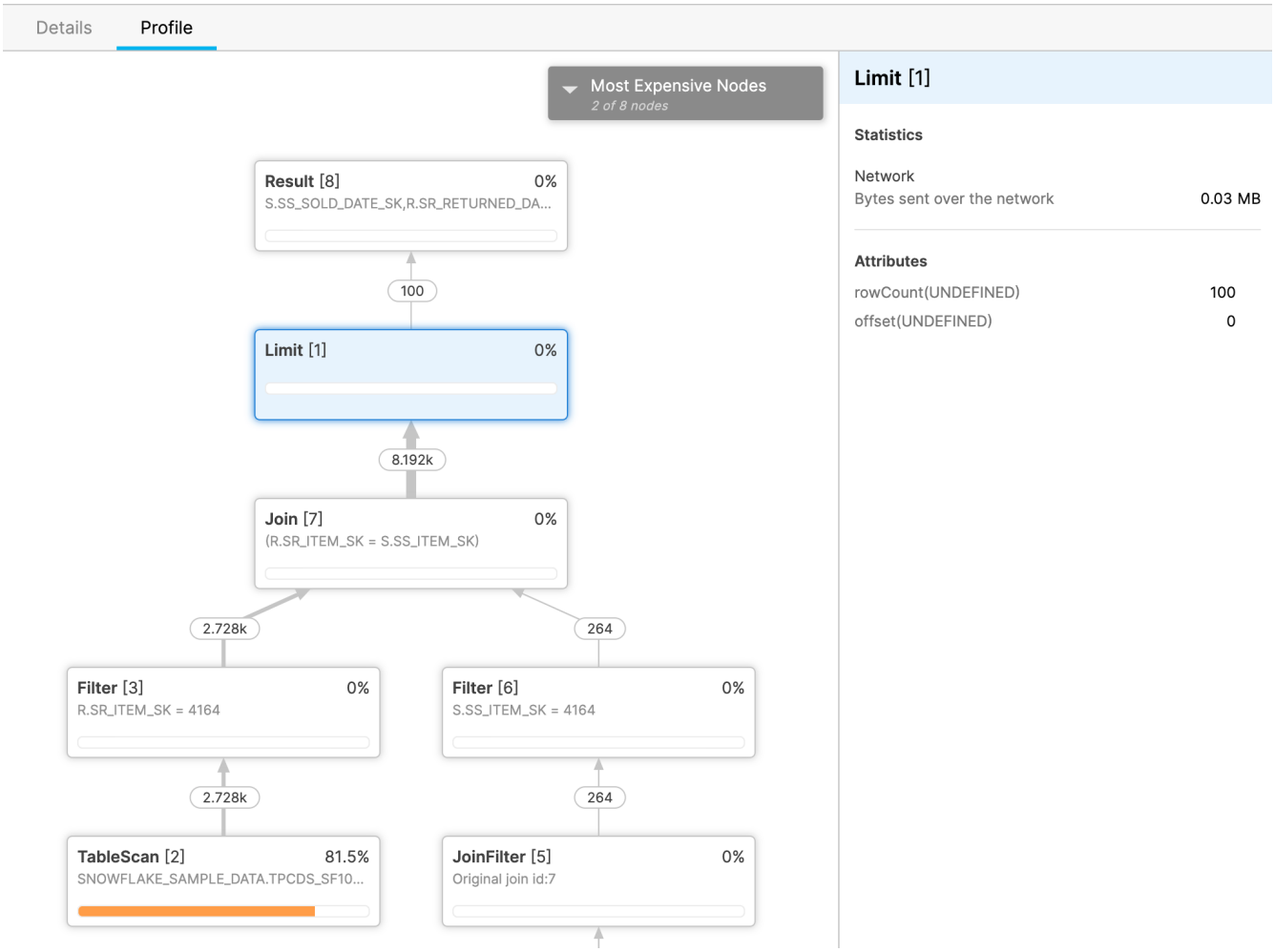

Figure 15: Limit Node

Notice that the LIMIT operator is processed at the very end of the query, and has no impact on table access or JOIN filtering. But the LIMIT clause does help to reduce the query result output, which helps to speed up the overall query performance.

11.4 Join Optimizations in Snowflake

JOIN is one of the most resource-intensive operations. The Snowflake optimizer provides built-in dynamic partition pruning to help reduce data access during join processing. If you use a JOIN filter column that is well-clustered, the query optimization can push down micro-partition pruning.

11.4.1 Set your context.

```
USE SCHEMA snowflake_sample_data.tpcds_sf10tcl;
```


11.4.2 Run the following query.

```
SELECT count(ss_customer_sk)
FROM store_sales JOIN date_dim d
ON ss_sold_date_sk = d_date_sk
WHERE d_year = 2000
GROUP BY ss_customer_sk;
```

11.4.3 Open the query profile, and click on the operator **TableScan [4]**.

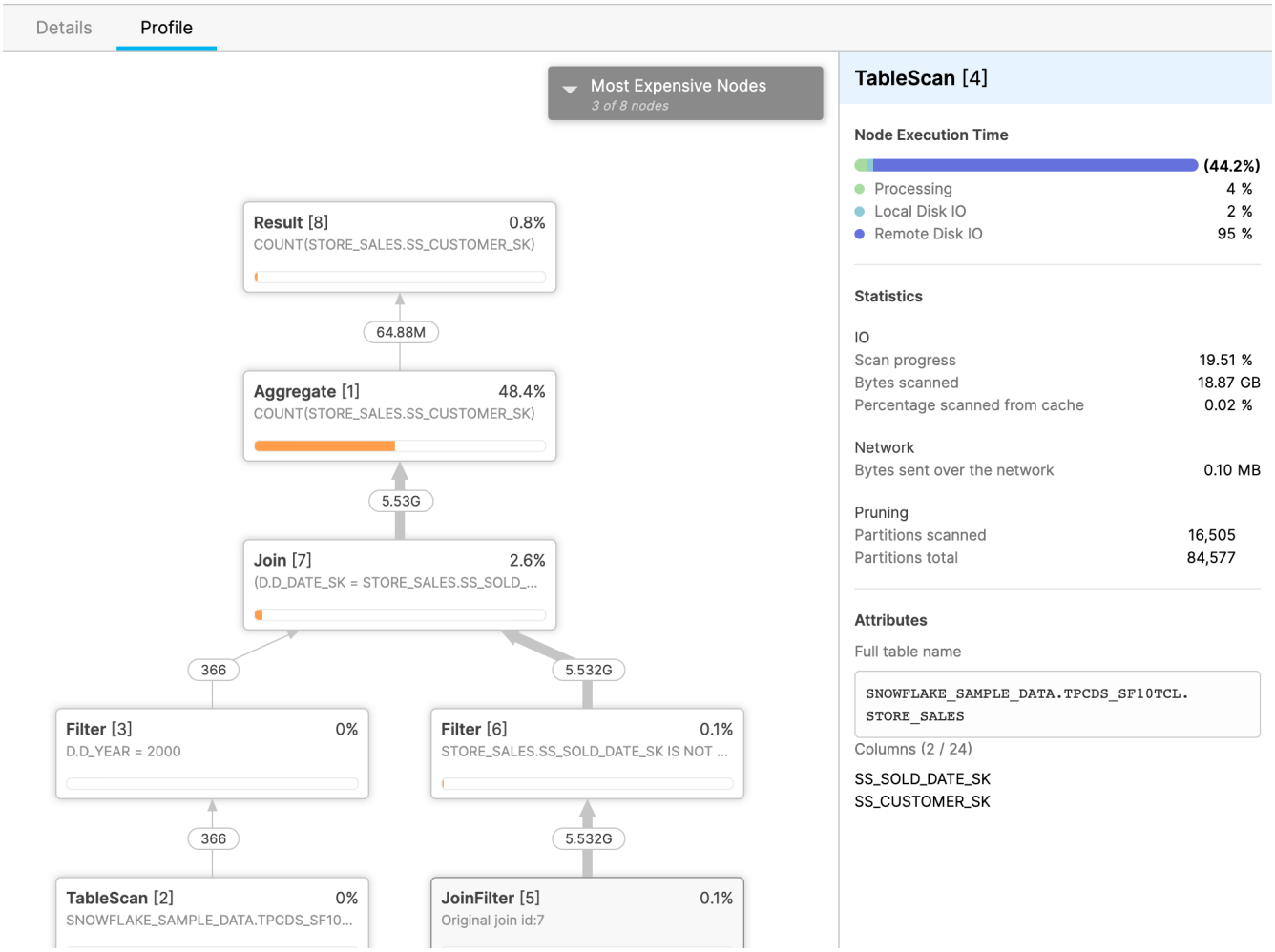

Figure 16: Table Scan Node

11.4.4 Take note of the performance metrics for this table scan.

The micro-partition pruning is fairly effective; the SQL pruner skipped a large number of micro-partitions. This corresponds to the filter: `D.D_DATE_SK = STORE_SALES.SS_SOLD_DATE_SK`

11.4.5 Check the clustering quality of the filter column.


```
SELECT SYSTEM$CLUSTERING_INFORMATION(
   'snowflake_sample_data.tpcds_sf10tcl.store_sales', '(ss_sold_date_sk)');
```

11.4.6 Review the result.

- The table is well clustered around the ss_sold_date_sk dimension
- The clustering depth is low
- The histogram shows most micro-partitions groups near the top

All of these contribute to better micro-partition pruning.

11.5 Using the Search Optimization Service

This lab takes you through the steps needed to identify a table that can benefit from a search optimization, and register the search optimization service on that table. You will perform a selective lookup query on the table both before and after applying the search optimization, and note the difference in performance.

11.5.1 Set your context.

```
CREATE WAREHOUSE IF NOT EXISTS [login]_QUERY_WH;
USE WAREHOUSE [login]_QUERY_WH;

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

CREATE SCHEMA IF NOT EXISTS SEARCH;
USE SCHEMA SEARCH;

ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

11.5.2 Create a table that can be enabled for the Search Optimization service.

Clone the web_sales table from training_db.traininglab. This allows you to put a search optimization on the cloned table, and compare performance of a point query between the cloned table and the original (wihtout the search optimization).

```
CREATE OR REPLACE TABLE WEB_SALES_SO CLONE TRAINING_DB.TRAININGLAB.WEB_SALES;
```

11.5.3 Register the cloned table to the search optimization service.

Search optimization is a table-level property and applies to all columns with supported data types (date, number, string, etc.).


```
ALTER TABLE WEB_SALES_SO ADD SEARCH OPTIMIZATION;
```

Search optimization has a maintenance service that runs in the background to create and maintain the search access paths. The service will check roughly once an hour for new work (table) that needs to have a search structure built.

Before you run a query to verify that search optimization is working, wait until this shows that the table has been fully optimized.

11.5.4 Verify that the search structure is complete.

Run the `SHOW TABLES` command to verify that search optimization has been added and to determine how much of the table has been optimized

```
SHOW TABLES LIKE '%WEB_SALES%';
SET QID = LAST_QUERY_ID();
DESCRIBE RESULT $QID;

SELECT
  "name", "search_optimization",
  "search_optimization_progress",
  "bytes" as "bytes in table",
  "search_optimization_bytes"
FROM TABLE(RESULT_SCAN($QID));
```

In the output from this command:

Verify that `SEARCH_OPTIMIZATION` is ON, which indicates that search optimization has been added.

Check the value of `SEARCH_OPTIMIZATION_PROGRESS`. This specifies the percentage of the table that has been optimized so far.

For example:

```
/*
  name        rows         bytes          search_optimization
     search_optimization_progress
  ----------- ------------ -------------- ---------------------
     ------------------------------
  WEB_SALES   7199963324   132571177472   ON                   100
*/
```

11.6 Test Search Optimization Performance

Search optimization works best to improve the performance of a query when the following conditions are true:

- The query requires a fast response (in seconds)
- Queries on the table frequently include an equality filter on columns that may not be well clustered
- The query does not include any expressions, functions, or conjunctions like OR

- At least one of the columns that is accessed through the query filter has at least 100k distinct values

11.6.1 Show order number, order date, and items sold for one customer on the original (not optimized) table:

```
SELECT
  WS_ORDER_NUMBER, WS_SOLD_DATE_SK, WS_ITEM_SK
FROM TRAINING_DB.TRAININGLAB.WEB_SALES
  WHERE
WS_BILL_CUSTOMER_SK = 956673;
```

11.6.2 Show order number, order date and items sold for one customer on the search-optimized table.

```
SELECT
  WS_ORDER_NUMBER, WS_SOLD_DATE_SK, WS_ITEM_SK
FROM WEB_SALES_SO
  WHERE
WS_BILL_CUSTOMER_SK = 956673;
```

11.6.3 Review the Query Profile and note the data access path is the Search Optimization Access operator.

11.6.4 Estimate the number of distinct values (cardinality) in the WS_BILL_CUSTOMER_SK column.

The point query executes fast in the previous step. This is because the cardinality of the filter column is sufficiently high:

```
SELECT
  APPROX_COUNT_DISTINCT(WS_BILL_CUSTOMER_SK)
FROM WEB_SALES_SO;
```

On the contrary, the column ws_web_site_sk has low cardinality:

```
SELECT
  APPROX_COUNT_DISTINCT(WS_WEB_SITE_SK)
FROM WEB_SALES_SO;
```

The query below using ws_web_site_sk as the search column is not a point query, so performance will not be helped by the search optimization.

```
SELECT
  WS_ORDER_NUMBER,WS_SOLD_DATE_SK, WS_ITEM_SK
FROM WEB_SALES_SO
  WHERE
WS_WEB_SITE_SK IN (1, 8, 23, 61, 73);
```


11.7 Explore Costs Associated with Search Optimization

These costs depend upon multiple factors, including the number of distinct values (NDVs) in the table. Typically, the size of the search optimization structure is approximately 25% of the original table's size.

In the extreme case where all columns have data types that use the search access path, and all data values in each column are unique, the required storage can be as much as the original table's size.

11.7.1 Find which columns have high cardinality (large number of distinct values)

Use the HLL function to approximate the count of distinct values in each column:;

```
SELECT
  HLL(WS_SOLD_DATE_SK),
  HLL(WS_SOLD_TIME_SK),
  HLL(WS_SHIP_DATE_SK),
  HLL(WS_ITEM_SK),
  HLL(WS_BILL_CUSTOMER_SK),
  HLL(WS_BILL_CDEMO_SK),
  HLL(WS_BILL_HDEMO_SK),
  HLL(WS_BILL_ADDR_SK),
  HLL(WS_SHIP_CUSTOMER_SK),
  HLL(WS_SHIP_CDEMO_SK),
  HLL(WS_SHIP_HDEMO_SK),
  HLL(WS_SHIP_ADDR_SK),
  HLL(WS_WEB_PAGE_SK),
  HLL(WS_WEB_SITE_SK),
  HLL(WS_SHIP_MODE_SK),
  HLL(WS_WAREHOUSE_SK),
  HLL(WS_PROMO_SK),
  HLL(WS_ORDER_NUMBER),
  HLL(WS_QUANTITY),
  HLL(WS_WHOLESALE_COST),
  HLL(WS_LIST_PRICE),
  HLL(WS_SALES_PRICE),
  HLL(WS_EXT_DISCOUNT_AMT),
  HLL(WS_EXT_SALES_PRICE),
  HLL(WS_EXT_WHOLESALE_COST)
FROM WEB_SALES_SO;
```

11.7.2 Note the cost of Search Optimization for building the structure on your big table

It is possible to use either the WebUI or SQL:

```
SELECT
  DATABASE_NAME,
  TABLE_ID,
  TABLE_NAME,
  START_TIME,
  END_TIME,
  CREDITS_USED
FROM SNOWFLAKE.ACCOUNT_USAGE.SEARCH_OPTIMIZATION_HISTORY
```



```
WHERE
  SCHEMA_NAME = CURRENT_SCHEMA()
  AND DATABASE_NAME = CURRENT_DATABASE();
```

11.7.3 Use the system function to estimate the costs of adding a search optimization to the table:

```
SELECT SYSTEM$ESTIMATE_SEARCH_OPTIMIZATION_COSTS('[login]_DB.SEARCH.WEB_SALES_SO');
```

12 Materialized View Use Cases

This lab will take approximately *15 minutes* to complete.

12.1 Cluster a Table Using a Timestamp Column

12.1.1 Set your context.

```
USE ROLE TRAINING_ROLE;
USE database [login]_db;
USE WAREHOUSE [login]_QUERY_WH;
ALTER SESSION SET USE_CACHED_RESULT=TRUE;
```

12.1.2 Create a table using cloning.

```
CREATE TABLE [login]_db.PUBLIC.weblog CLONE TRAINING_DB.TRAININGLAB.WEBLOG;
```

12.1.3 Check the clustering quality of the CREATE_MS and METRIC9 columns.

```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'WEBLOG' , '(CREATE_MS)');

SELECT SYSTEM$CLUSTERING_INFORMATION( 'WEBLOG' , '(METRIC9)');
```

Which column is more effectively clustered?

12.1.4 Run a query with a search filter using the column CREATE_MS.

```
SELECT COUNT(*) CNT
     , AVG(TIME_ON_LOAD_MS) AVG_TIME_ON_LOAD
FROM WEBLOG
WHERE CREATE_MS BETWEEN 1000000000 AND 1000001000;
```


12.1.5 View the query profile to check micro-partition pruning.

In this case, the micro-partition pruning is very good.

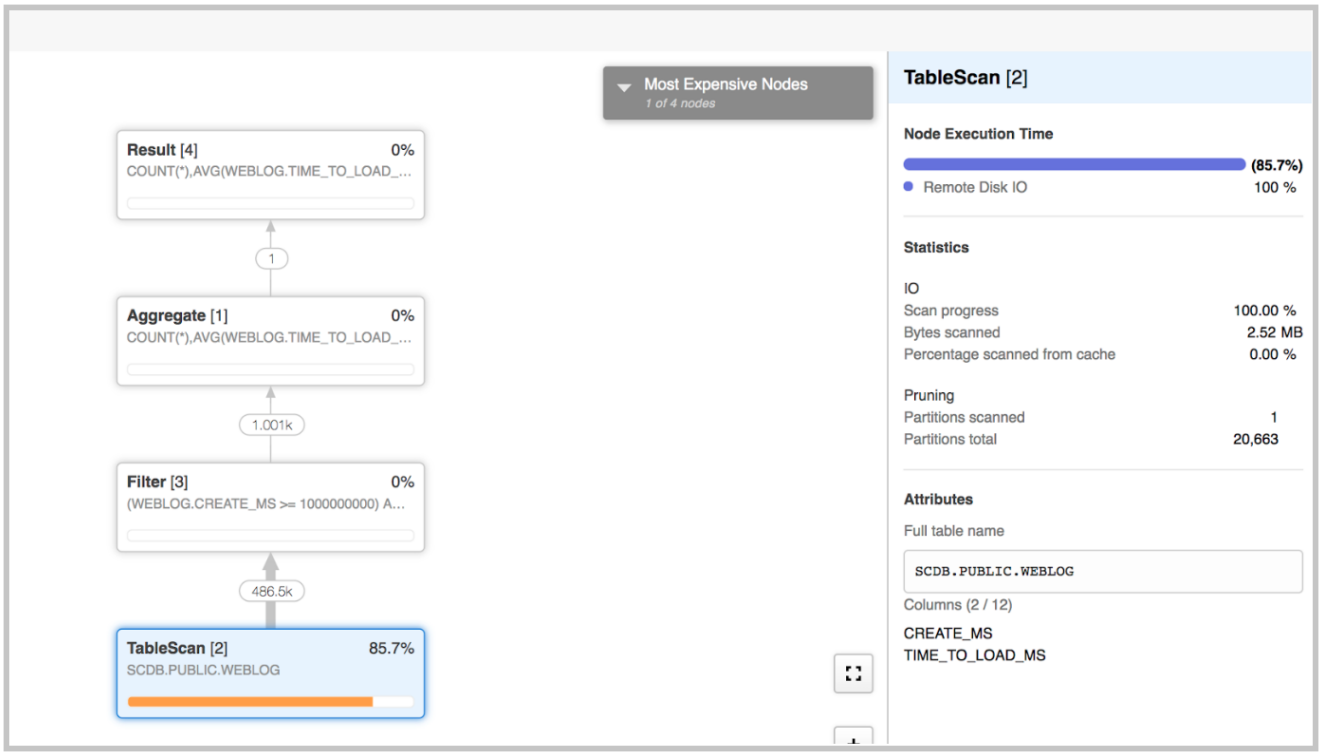

Figure 17: TableScan[2] Node Details

12.1.6 Check the clustering quality of the column PAGE_ID.

Based on the column name - would you expect it to be well-clustered, or poorly clustered?

```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'WEBLOG' , '(PAGE_ID)' );
```

Were you right?

12.1.7 Run a query that filters in the PAGE_ID.

Since PAGE_ID is not well-clustered, you would expect the micro-partition pruning to be low.

```
SELECT COUNT(*) CNT
     , AVG(TIME_ON_LOAD_MS) AVG_TIME_ON_LOAD
FROM WEBLOG
WHERE PAGE_ID=100000;
```


12.1.8 Check the micro-partition pruning in the query profile.

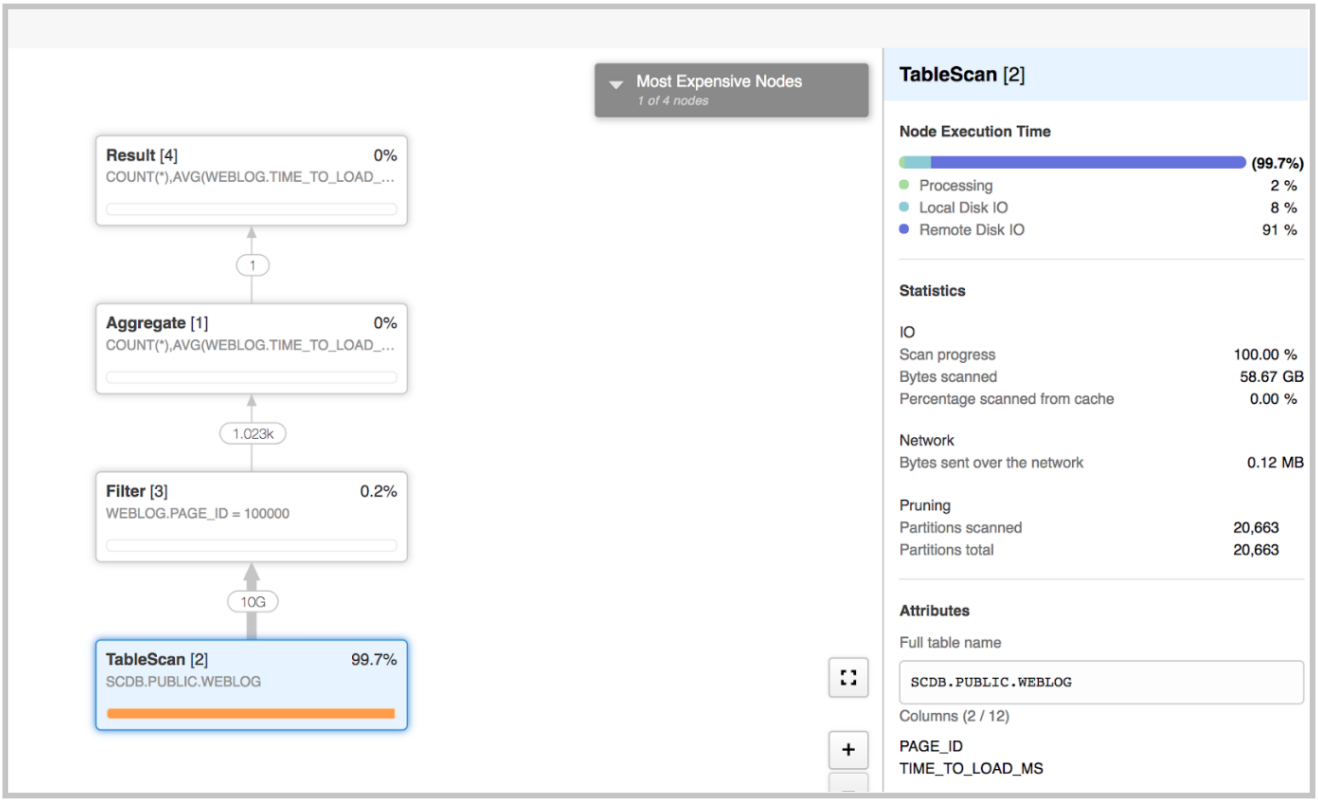

Figure 18: TableScan[2] Node Details

Note that, as expected, the micro-partition pruning is very low. Record the execution time, and the micro-partition pruning.

12.2 Cluster a Table to Improve Query Performance

You would like both queries - the one filtered by PAGE_ID and the one filtered by CREATE_MS - to run fast. But running both queries with equally good performance requires using a second copy of the data that's organized differently. You can do this easily with materialized views.

12.2.1 Create a materialized view clustered by PAGE_ID.

Creating the materialized view with a clustering key causes Snowflake to reorganize the data during the initial creation of the materialized view. Here you will increase the virtual warehouse size so the re-clustering will go faster - but the operation will still take up to 10 minutes. This would be a good time to stretch your legs or refill your coffee.

```
ALTER WAREHOUSE [login]_query_wh set warehouse_size=XLARGE;
```



```
CREATE OR REPLACE MATERIALIZED VIEW MV_TIME_ON_LOAD
    (CREATE_MS,
    PAGE_ID,
    TIME_ON_LOAD_MS)
    CLUSTER BY (PAGE_ID)
AS
SELECT
    CREATE_MS,
    PAGE_ID,
    TIME_ON_LOAD_MS
FROM WEBLOG;
```

12.2.2 Check clustering efficiency on the PAGE_ID column of the materialized view.

```
SELECT SYSTEM$CLUSTERING_INFORMATION ( 'MV_TIME_ON_LOAD' , '(PAGE_ID)' );
```

After the clustering, the average_depth should be around 2 or 3. This is quite an improvement.

12.2.3 Run the query filtered on PAGE_ID against the materialize view.

For a proper performance comparison, set the warehouse size back to what it was the first time the query ran.

```
ALTER WAREHOUSE [login]_query_wh set warehouse_size=LARGE;

SELECT COUNT(*), AVG(TIME_ON_LOAD_MS) AVG_TIME_ON_LOAD
FROM MV_TIME_ON_LOAD
WHERE PAGE_ID=100000;
```

This example illustrates a substantial improvement in terms of query performance.

12.2.4 Check micro-partition pruning in the query profile.

With the materialized view, only one micro-partition was scanned. There was a significant increase in performance as a result.

12.2.5 SHOW materialized views on the WEBLOG table.

```
SHOW MATERIALIZED VIEWS ON weblog;
```

12.3 Explore Automatic Transparent Rewrite on Materialized Views

The Snowflake query optimizer can exploit materialized views to automatically rewrite/reroute queries made against the source table, to the materialized view.

12.3.1 Use EXPLAIN to see if a command will use a source table or a materialized view.

Use explain to check if a query against the original source table will use a materialized view for query performance

```
EXPLAIN
SELECT COUNT(*) CNT
     , AVG(TIME_ON_LOAD_MS) AVG_TIME_ON_LOAD
FROM WEBLOG
WHERE PAGE_ID=100000;
```

Note that even though the query was against the WEBLOG table, the EXPLAIN plan shows that the materialized view will be scanned.

12.3.2 Run the query.

```
ALTER SESSION SET USE_CACHED_RESULT = FALSE;

SELECT COUNT(*) CNT
     , AVG(TIME_ON_LOAD_MS) AVG_TIME_ON_LOAD
FROM WEBLOG
WHERE PAGE_ID=100000;
```

12.3.3 Check the query profile.

Even though the query was against the WEBLOG table, the materialized view was scanned instead.

12.4 Materialized Views on External Tables

12.4.1 Create a file format for an external table.

```
create or replace file format txt_fixed_width
  TYPE = CSV
  COMPRESSION = 'AUTO'
  FIELD_DELIMITER = NONE
  RECORD_DELIMITER = '\\n'
  SKIP_HEADER = 0
  TRIM_SPACE = FALSE
  ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE
  NULL_IF = ('\\N');
```

12.4.2 Create an external table with partitions based on the filename.

```
create or replace external table finwire
(
    YEAR  VARCHAR(4) AS SUBSTR(METADATA$FILENAME, 16, 4)
  , QUARTER VARCHAR(1) AS SUBSTR(METADATA$FILENAME, 21, 1)
```



```
  , thestring varchar(90) AS  SUBSTR(METADATA$FILENAME, 1, 50)
  , PTS VARCHAR(15) AS SUBSTR($1, 8, 15)
  , REC_TYPE VARCHAR(3) AS SUBSTR($1, 23, 3)
  , COMPANY_NAME VARCHAR(60) AS SUBSTR($1, 26, 60)
  , CIK VARCHAR(10) AS SUBSTR($1, 86, 10)
  , STATUS VARCHAR(4) AS IFF(SUBSTR($1, 23, 3) = 'CMP', SUBSTR($1, 96,
     4),SUBSTR($1, 47, 4))
  , INDUSTRY_ID VARCHAR(2) AS SUBSTR($1, 100, 2)
  , SP_RATING VARCHAR(4) AS SUBSTR($1, 102, 4)
  , FOUNDING_DATE VARCHAR(8) AS SUBSTR($1, 106, 8)
  , ADDR_LINE1 VARCHAR(80) AS SUBSTR($1, 114, 80)
  , ADDR_LINE2 VARCHAR(80) AS SUBSTR($1, 194, 80)
  , POSTAL_CODE VARCHAR(12) AS SUBSTR($1, 274, 12)
  , CITY VARCHAR(25) AS SUBSTR($1, 286, 25)
  , STATE_PROVINCE VARCHAR(20) AS SUBSTR($1, 311, 20)
  , COUNTRY VARCHAR(24) AS SUBSTR($1, 331, 24)
  , CEO_NAME VARCHAR(46) AS SUBSTR($1, 355, 46)
  , DESCRIPTION VARCHAR(150) AS SUBSTR($1, 401, 150)
  , QTR_START_DATE VARCHAR(8) AS SUBSTR($1, 31, 8)
  , POSTING_DATE VARCHAR(8) AS SUBSTR($1, 39, 8)
  , REVENUE VARCHAR(17) AS SUBSTR($1, 47, 17)
  , EARNINGS VARCHAR(17) AS SUBSTR($1, 64, 17)
  , EPS VARCHAR(12) AS SUBSTR($1, 81, 12)
  , DILUTED_EPS VARCHAR(12) AS SUBSTR($1, 93, 12)
  , MARGIN VARCHAR(12) AS SUBSTR($1, 105, 12)
  , INVENTORY VARCHAR(17) AS SUBSTR($1, 117, 17)
  , ASSETS VARCHAR(17) AS SUBSTR($1, 134, 17)
  , LIABILITIES VARCHAR(17) AS SUBSTR($1, 151, 17)
  , SH_OUT VARCHAR(13)AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 168, 13),
    SUBSTR($1, 127, 13))
  , DILUTED_SH_OUT VARCHAR(13) AS SUBSTR($1, 181, 13)
  , CO_NAME_OR_CIK VARCHAR(60) AS IFF(SUBSTR($1, 23, 3) = 'FIN', SUBSTR($1, 194,
     10), SUBSTR($1, 168, 10))
  , SYMBOL VARCHAR(15) AS SUBSTR($1, 26, 15)
  , ISSUE_TYPE VARCHAR(6) AS SUBSTR($1, 41, 6)
  , NAME VARCHAR(70) AS SUBSTR($1, 51, 70)
  , EX_ID VARCHAR(6) AS SUBSTR($1, 121, 6)
  , FIRST_TRADE_DATE VARCHAR(8) AS SUBSTR($1, 140, 8)
  , FIRST_TRADE_EXCHG VARCHAR(8) AS SUBSTR($1, 148, 8)
  , DIVIDEND VARCHAR(12) AS SUBSTR($1, 156, 12)
)
partition by (year,quarter)
location = @training_db.traininglab.ed_stage/finwire
file_format = (format_name = 'txt_fixed_width');
```

12.4.3 Refresh the external table.

```
ALTER EXTERNAL TABLE finwire REFRESH;
```

12.4.4 Execute some queries and examine their profiles.

```
select co_name_or_cik
```



```
    , year
    , quarter
    , sum(revenue::number)
  from finwire
  where rec_type='FIN' and year='1967' and quarter='3' group by 1,2,3;

select co_name_or_cik
    , year
    , quarter
    , sum(revenue::number)
  from finwire
  where rec_type='FIN' and year='1989' and quarter='3' group by 1,2,3;
```

12.4.5 Create a materialized view that filters on REC_TYPE = 'CMP'.

```
CREATE OR REPLACE MATERIALIZED VIEW FINWIRE_CMP AS
 SELECT TO_TIMESTAMP_NTZ(PTS,'YYYYMMDD-HH24MISS') AS PTS
      , REC_TYPE
      , COMPANY_NAME
      , CIK, STATUS
      , INDUSTRY_ID
      , SP_RATING
      , TRY_TO_DATE(FOUNDING_DATE) AS FOUNDING_DATE
      , ADDR_LINE1
      , ADDR_LINE2
      , POSTAL_CODE
      , CITY
      , STATE_PROVINCE
      , COUNTRY
      , CEO_NAME
      , DESCRIPTION
    FROM FINWIRE WHERE REC_TYPE = 'CMP';
```

12.4.6 Create a materialized view that filters on REC_TYPE = 'FIN'.

```
CREATE OR REPLACE MATERIALIZED VIEW FINWIRE_FIN AS
  SELECT TO_TIMESTAMP_NTZ(PTS,'YYYYMMDD-HH24MISS') AS PTS,
  REC_TYPE, TO_NUMBER(YEAR,4,0) AS YEAR,
  TO_NUMBER(QUARTER,1,0) AS QUARTER,
  TO_DATE(QTR_START_DATE, 'YYYYMMDD') AS QTR_START_DATE,
  TO_DATE(POSTING_DATE, 'YYYYMMDD') AS POSTING_DATE,
  TO_NUMBER(REVENUE,15,2) AS REVENUE,
  TO_NUMBER(EARNINGS,15,2) AS EARNINGS,
  TO_NUMBER(EPS,10,2) AS EPS,
  TO_NUMBER(DILUTED_EPS,10,2) AS DILUTED_EPS,
  TO_NUMBER(MARGIN,10,2) AS MARGIN,
  TO_NUMBER(INVENTORY,15,2) AS INVENTORY,
  TO_NUMBER(ASSETS,15,2) AS ASSETS,
  TO_NUMBER(LIABILITIES,15,2) AS LIABILITIES,
  TO_NUMBER(SH_OUT,13,0) AS SH_OUT,
  TO_NUMBER(DILUTED_SH_OUT,13,0) AS DILUTED_SH_OUT,
  CO_NAME_OR_CIK
```



```
FROM FINWIRE WHERE REC_TYPE = 'FIN';
```

12.4.7 Create a materialized view that filters on REC_TYPE = ‘SEC’

.

```
CREATE OR REPLACE MATERIALIZED VIEW FINWIRE_SEC AS SELECT
TO_TIMESTAMP_NTZ(PTS,'YYYYMMDD-HH24MISS') AS PTS, REC_TYPE, SYMBOL, ISSUE_TYPE,
    STATUS, NAME, EX_ID,
TO_NUMBER(SH_OUT,13,0) AS SH_OUT, TO_DATE(FIRST_TRADE_DATE,'YYYYMMDD') AS
    FIRST_TRADE_DATE,
TO_DATE(FIRST_TRADE_EXCHG,'YYYYMMDD') AS FIRST_TRADE_EXCHG,
    TO_NUMBER(DIVIDEND,10,2) AS DIVIDEND, CO_NAME_OR_CIK
FROM FINWIRE WHERE REC_TYPE = 'SEC';
```

12.4.8 SHOW the materialized views.

```
SHOW MATERIALIZED VIEWS;
```

12.4.9 Run an query on the FINWIRE_FIN materialized view.

```
select co_name_or_cik, year, quarter, sum(revenue)
  from finwire_fin
  where rec_type='FIN' and year=1967 and quarter=2 group by 1,2,3;
```

12.4.10 View the query profile.

13 Three Vectors of Scaling a Virtual Warehouse

This lab will take approximately *20 minutes* to complete.

13.1 Create Warehouses for the Lab

13.1.1 Create a LOAD virtual warehouse:

```
USE ROLE TRAINING_ROLE;

CREATE OR REPLACE WAREHOUSE [login]_LOAD_WH
  WAREHOUSE_SIZE = 'LARGE'
  WAREHOUSE_TYPE = 'STANDARD'
  AUTO_SUSPEND = 65
  AUTO_RESUME = true
  MIN_CLUSTER_COUNT = 1
  MAX_CLUSTER_COUNT = 2
```



```
  INITIALLY_SUSPENDED = true
  SCALING_POLICY = 'STANDARD'
  COMMENT = 'Training WH for completing hands-on lab queries';
```

13.1.2 Create a QUERY virtual warehouse.

```
CREATE OR REPLACE WAREHOUSE [login]_QUERY_WH
  WAREHOUSE_SIZE = 'LARGE'
  WAREHOUSE_TYPE = 'STANDARD'
  AUTO_SUSPEND = 60
  AUTO_RESUME = true
  MIN_CLUSTER_COUNT = 1
  MAX_CLUSTER_COUNT = 2
  INITIALLY_SUSPENDED = true
  SCALING_POLICY = 'STANDARD'
  COMMENT = 'Training WH for completing hands-on lab queries';
```

13.1.3 `SHOW` the warehouses.

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

13.2 Explore AUTO_SUSPEND and AUTO_RESUME

13.2.1 Set the auto_suspend time on your LOAD warehouse to one minute.

```
ALTER WAREHOUSE [login]_LOAD_WH SET AUTO_SUSPEND = 60;
```

13.2.2 Verify your warehouse using the top ribbon.

In the top ribbon, select the `Warehouses` tab. Locate your warehouse and confirm that the auto_suspend time is set to 1 minutes. Then return to your worksheet.

13.2.3 Suspend your `LOAD` warehouse.

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

If you get the message, “Invalid State. Warehouse `[login]_LOAD_WH` cannot be suspended”, that means that the warehouse is already suspended. You can ignore this message whenever it appears.

13.2.4 Locate your warehouse in the top ribbon and confirm that it is now suspended.

When you have verified its state, return to the worksheet.

13.2.5 Submit a query on a suspended warehouse.

Auto_resume is enabled by default when you create a virtual warehouse. When you submit a command that requires a warehouse, it will automatically resume.

```
USE WAREHOUSE [login]_QUERY_WH;

--Turn off the query result cache so the warehouse must be used for the query.
ALTER SESSION SET USE_CACHED_RESULT = false;

SELECT * FROM TRAINING_DB.TRAININGLAB.REGION;
```

Because the warehouse is configured to auto resume, it resumed automatically when it was needed to process a query.

13.2.6 Locate your warehouse in the top ribbon and confirm that it is now running.

When you have verified its state, return to the worksheet.

13.3 Size Warehouses Up and Down

Resizing can be completed at any time, even when the virtual warehouse is running.

13.3.1 Resize one of your warehouses.

```
ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE = 'X-LARGE';

SHOW WAREHOUSES LIKE '[login]_QUERY_WH';

ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE = 'MEDIUM';

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

Note the change in each warehouse size after running the ALTER commands.

13.3.2 Locate your warehouse in the top ribbon and confirm that its size is now MEDIUM.

Return to the worksheet when you are done.

13.3.3 Set your virtual warehouse size to LARGE.

```
USE ROLE training_role;
USE WAREHOUSE [login]_query_wh;
USE DATABASE snowflake_sample_data;
USE SCHEMA TPCDS_SF10TCL;

ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE=LARGE;

ALTER SESSION SET USE_CACHED_RESULT=FALSE;
```

13.3.4 Run query 3 of the TPCDS_SF10TCL schema.

```
SELECT dt.d_year
  ,item.i_brand_id brand_id
  ,item.i_brand brand
  ,sum(ss_net_profit) sum_agg
FROM  date_dim dt
  ,store_sales
  ,item
WHERE dt.d_date_sk = store_sales.ss_sold_date_sk
  and store_sales.ss_item_sk = item.i_item_sk
  and item.i_manufact_id = 939
  and dt.d_moy=12
GROUP BY dt.d_year
  ,item.i_brand
  ,item.i_brand_id
ORDER BY dt.d_year
  ,sum_agg desc
  ,brand_id
LIMIT 100;
```


13.3.5 Examine the query profile overview and record the execution time.

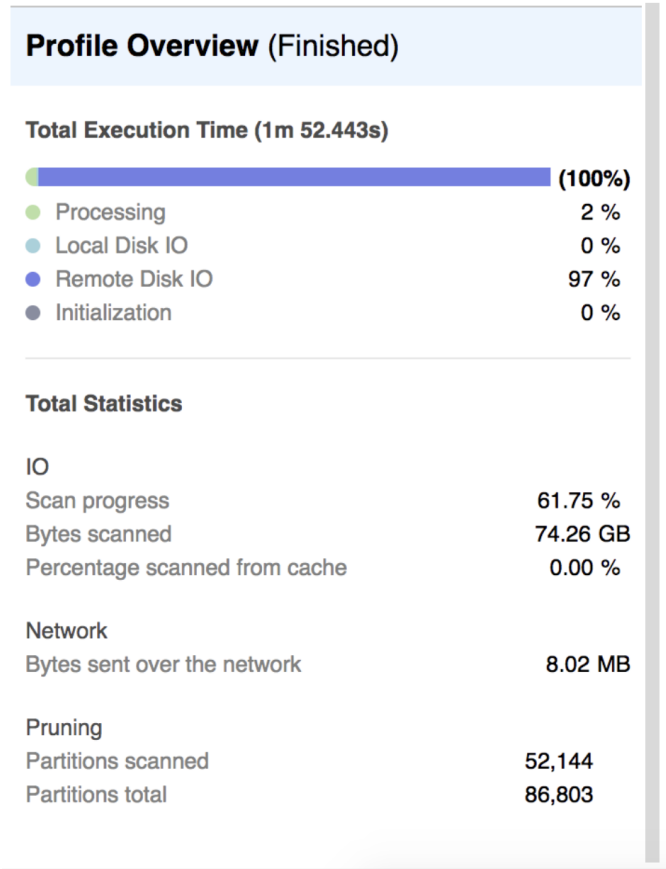

Figure 19: Query Profile (Large virtual warehouse)

13.3.6 Set your virtual warehouse size to XLARGE.

```
ALTER WAREHOUSE [login]_QUERY_WH SUSPEND;

ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE=XLARGE;
```

13.3.7 Re-run query 3 of the TPCDS_SF10TCL schema.

```
SELECT dt.d_year
  ,item.i_brand_id brand_id
  ,item.i_brand brand
  ,sum(ss_net_profit) sum_agg
FROM  date_dim dt
  ,store_sales
  ,item
WHERE dt.d_date_sk = store_sales.ss_sold_date_sk
  and store_sales.ss_item_sk = item.i_item_sk
  and item.i_manufact_id = 939
  and dt.d_moy=12
```



```
GROUP BY dt.d_year
  ,item.i_brand
  ,item.i_brand_id
ORDER BY dt.d_year
  ,sum_agg desc
  ,brand_id
LIMIT 100;
```

13.3.8 Examine the query profile overview and record the execution time.

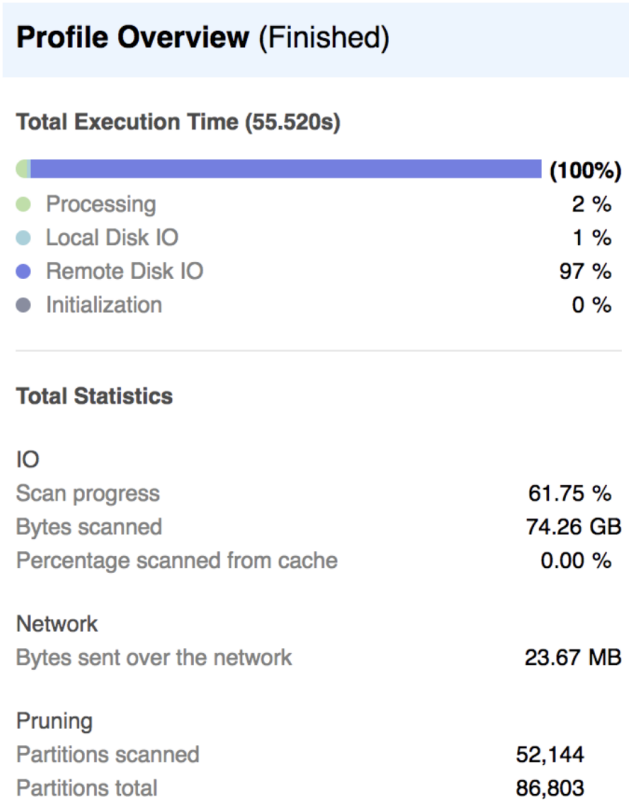

Figure 20: Query Profile (X-Large virtual warehouse)

Observation: Doubling the virtual warehouse size cut the elapsed time in half. The query ran twice as fast, for the same cost (as billed per second).

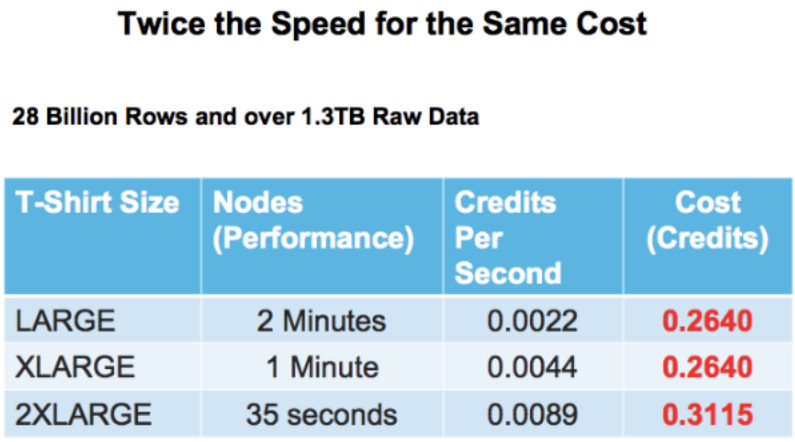

Figure 21: Twice the speed for the same cost

13.3.9 Set the warehouse size back to LARGE, and turn on the query result cache.

```
ALTER WAREHOUSE [login]_QUERY_WH SET WAREHOUSE_SIZE=LARGE;
ALTER SESSION SET USE_CACHED_RESULT=TRUE;
```

13.4 Size Warehouses Out and Back

A Snowflake multi-cluster warehouse consists of one or more clusters of servers that execute queries. For a given warehouse, a user can set both the minimum and maximum number of compute clusters to allocate to that warehouse.

13.4.1 Create an auto-scaling virtual warehouse.

```
CREATE OR REPLACE WAREHOUSE [login]_SCALE_OUT_WH
WAREHOUSE_SIZE = 'SMALL'
  AUTO_SUSPEND = 60
  AUTO_RESUME = true
  MIN_CLUSTER_COUNT = 1
  MAX_CLUSTER_COUNT = 3
  INITIALLY_SUSPENDED = true
  SCALING_POLICY = 'STANDARD'
  COMMENT = 'Training WH for completing concurrency tests';

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

Note: By setting the `MAX_CLUSTER_COUNT` greater than the `MIN_CLUSTER_COUNT`, you are configuring the Warehouse in auto-scale mode. This allows Snowflake to scale the warehouse as needed to handing fluctuating workloads.

14 Performance Analysis Toolkit and Tuning Metrics

This lab provides examples for identifying some of the most common performance bottlenecks and issues you may encounter when running queries in Snowflake. This set of exercises also shows the set of SQL operations that are most commonly associated with these performance bottlenecks and issues.

14.1 Explore Spilling to Local and Remote Storage

14.1.1 Set your context, and turn off the query result cache.

```
USE ROLE training_role;
USE WAREHOUSE [login]_query_wh;
USE database [login]_db;
CREATE SCHEMA PERFORMANCE_LAB;
USE SCHEMA snowflake_sample_data.tpcds_sf10tcl;
ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

14.1.2 Resize your warehouse to SMALL.

```
ALTER WAREHOUSE [login]_query_wh SUSPEND;
ALTER WAREHOUSE [login]_query_wh SET WAREHOUSE_SIZE = SMALL;
ALTER WAREHOUSE [login]_query_wh RESUME;
```

14.1.3 Run a query with a window function on the small warehouse.

The query lists detailed catalog sales data together with a running sum of sales price within the order. On an small warehouse, this will take a few minutes to complete.

```
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) running_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;
```

14.1.4 Open the query profile and select the operator **WindowFunction [1]**

Note how long the query took, and how much of the query time was spent on the window function. Also note that your query is spilling to local storage, and possibly to remote storage.

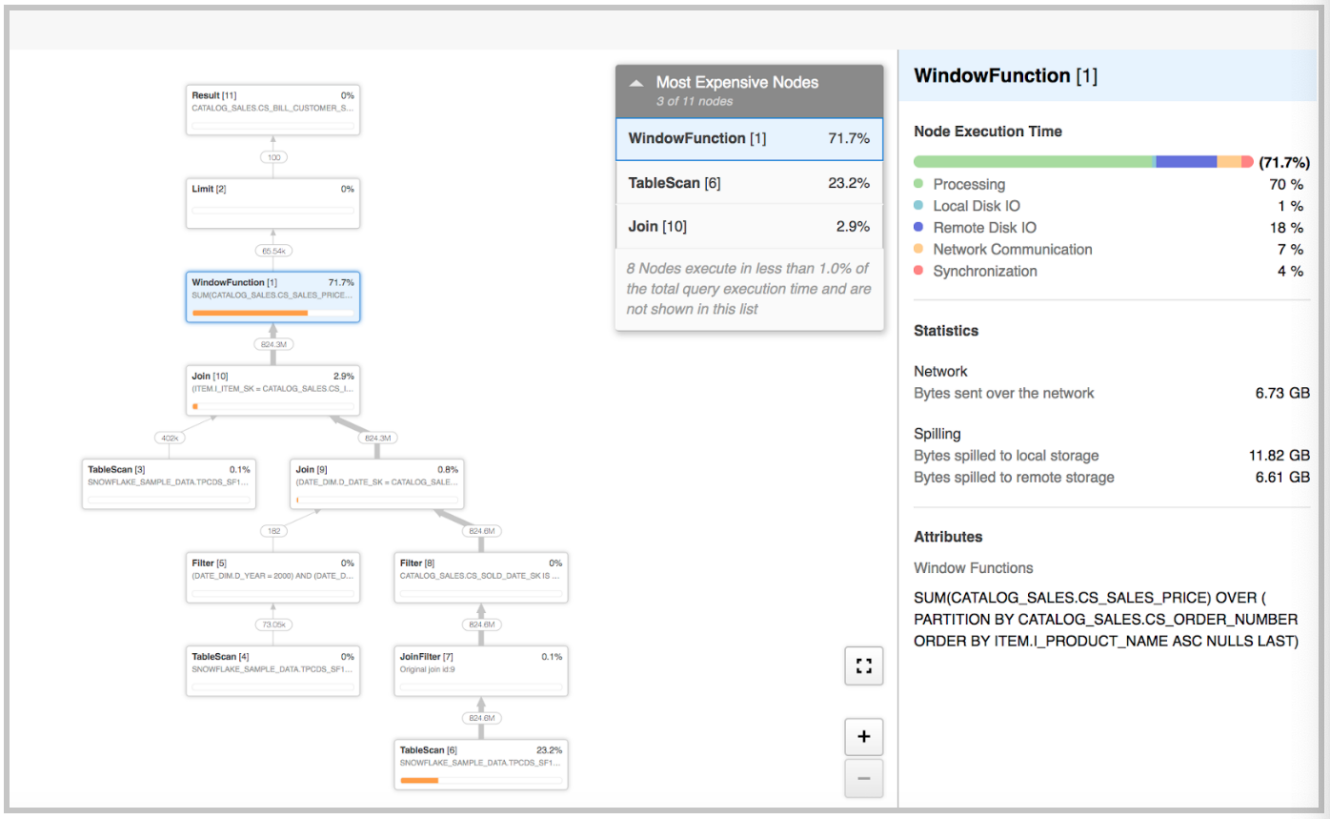

Figure 22: Query Profile Small Warehouse Spilling

14.1.5 Resize your warehouse to MEDIUM.

```
ALTER WAREHOUSE [login]_query_wh SUSPEND;
ALTER WAREHOUSE [login]_query_wh SET WAREHOUSE_SIZE = MEDIUM;
ALTER WAREHOUSE [login]_query_wh RESUME;
```

14.1.6 Re-run the query.

```
SELECT cs_bill_customer_sk
     , cs_order_number
     , i_product_name
     , cs_sales_price
     , SUM(cs_sales_price) OVER (PARTITION BY cs_order_number
        ORDER BY i_product_name
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) running_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;
```


14.1.7 Open the query profile.

What was the query time? Click on the WindowFunction node and see if the query is still spilling to local or remote storage.

14.1.8 Resize your virtual warehouse to LARGE.

```
ALTER WAREHOUSE [login]_query_wh SUSPEND;
ALTER WAREHOUSE [login]_query_wh SET WAREHOUSE_SIZE = LARGE;
ALTER WAREHOUSE [login]_query_wh RESUME;
```

14.1.9 Re-run the query.

```
SELECT cs_bill_customer_sk
     , cs_order_number
     , i_product_name
     , cs_sales_price
     , SUM(cs_sales_price) OVER (PARTITION BY cs_order_number
        ORDER BY i_product_name
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) running_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;
```

14.1.10 View the query profile.

What was the execution time? Is the WindowFunction still spilling?

Snowflake's architecture allows for linear scaling of compute resources, enabling both better performance and lower cost at the same time!

```
/*
Cluster Size   Node Count   Response Time   Credit Cost
-------------- ------------ --------------- -------------
Small          2            3 mins          0.108
Medium         4            1 min 20s       0.088
Large          8            30s             0.066
*/
```

14.2 Evaluate WHERE Clauses Based on Clustering Efficiency

In this scenario, we look into large table scan symptoms caused by the fact that the query's FILTER column is not the clustering dimension of the base table

14.2.1 Set your context and warehouse size.

```
USE SCHEMA training_db.tpch_sf1000;

ALTER SESSION SET USE_CACHED_RESULT = false;
ALTER WAREHOUSE [login]_query_wh SUSPEND;
ALTER WAREHOUSE [login]_query_wh SET WAREHOUSE_SIZE = MEDIUM;
ALTER WAREHOUSE [login]_query_wh RESUME;
```

14.2.2 Run a query with a filter on a column that is not well-clustered.

```
SELECT COUNT(*)
FROM  lineitem
WHERE l_extendedprice < 30000;
```

14.2.3 View the query profile and note the micro-partition pruning.

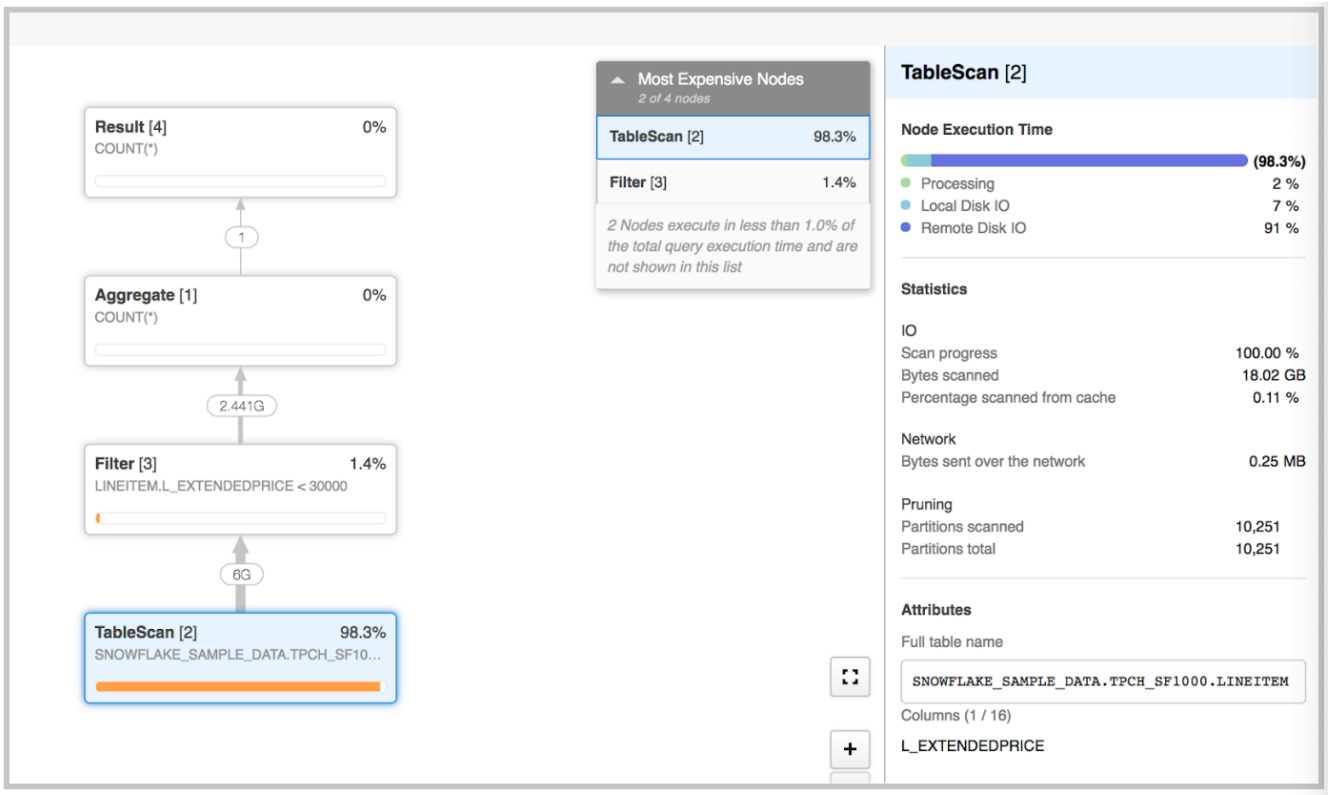

Figure 23: Query Profile Medium Warehouse

Note that all micro-partitions were scanned.

14.2.4 Evalute the clustering efficiency on the l_extendedprice column.


```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'lineitem' , '(l_extendedprice)' );
```

Note that the table is poorly clustered on the l_extendedprice dimension. The clustering depth shows almost 100% overlap in the micro-paritions.

14.2.5 Run a query that filters on a well-clustered column.

```
SELECT COUNT(*)
  FROM lineitem
  WHERE l_shipdate > to_date('1995-03-15');
```

14.2.6 Open the query profile and view micro-partition pruning.

14.2.7 Evaluate the clustering efficiency of the filter column.

```
SELECT SYSTEM$CLUSTERING_INFORMATION( 'lineitem' ,   '(l_shipdate)' );
```

Summary:
The amount of data scanned by a query is directly related to how well a query's WHERE clause column correlates to the clustering dimension of the table being accessed.

14.3 Rogue Query Symptom from JOIN Explosion

In this scenario, we will explore run away query symptoms which are common in real workloads. One common type of rogue query includes join pitfalls like explosion of output and unintentional cross joins.

14.3.1 Set your context.

```
USE SCHEMA  snowflake_sample_data.tpcds_sf100tcl;
ALTER WAREHOUSE [login]_query_wh SET WAREHOUSE_SIZE = medium;
USE WAREHOUSE [login]_query_wh;
```

14.3.2 Run the following query:

This may take 3-4 minutes to complete.

```
SELECT S.SS_SOLD_DATE_SK
     , R.SR_RETURNED_DATE_SK
     , S.SS_STORE_SK
     , S.SS_ITEM_SK
     , S.SS_SALES_PRICE
     , S.SS_SALES_PRICE
     , R.SR_RETURN_AMT
  FROM STORE_SALES  S
```



```
    INNER JOIN STORE_RETURNS  R
       on R.SR_ITEM_SK=S.SS_ITEM_SK
 WHERE  S.SS_ITEM_SK =4164;
```

14.3.3 Review the Query Profile.

Note that the JOIN produces a large number of output records relative to the sizes of the two input data sets.

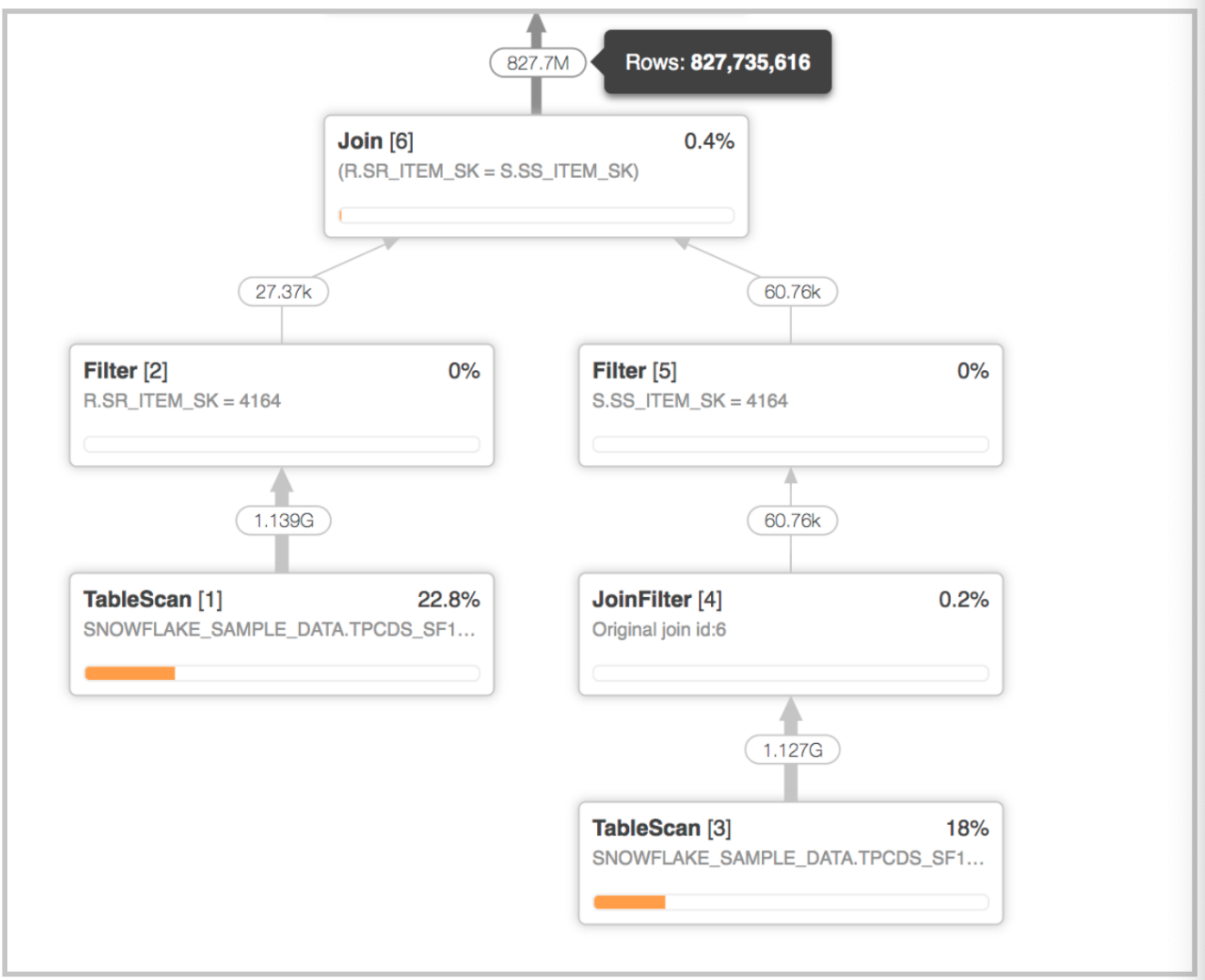

Figure 24: Query Profile Explode Join

To avoid performance issues with a JOIN that could explode the outputs, you should JOIN on unique keys. If it's not possible to join on unique keys, consider other options such as adding more filters to the WHERE clause.

14.4 Tune Timeout Parameters

In this exercise, we will explore tuning the timeout parameter available to a virtual warehouse to manage long-running workloads.

14.4.1 Review the existing values of the timeout parameters:

```
SHOW PARAMETERS IN WAREHOUSE [login]_query_wh;
```

Row	key	value	default	level	description	type
1	MAX_CONCURRENCY_LEVEL	8	8		Maximum number of S...	NUMBER
2	STATEMENT_QUEUED_TIMEOUT_IN_SECONDS	0	0		Timeout in seconds for...	NUMBER
3	STATEMENT_TIMEOUT_IN_SECONDS	172800	172800		Timeout in seconds for...	NUMBER

Figure 25: Show Warehouse Results

You will see that the maximum timeout for a queued statement is 0 seconds (meaning it will never time out). The maximum timeout of a statement is 2 days.

14.4.2 Update the statement_timeout_in_seconds parameter to 30 seconds.

```
ALTER WAREHOUSE [login]_query_wh SET STATEMENT_TIMEOUT_IN_SECONDS = 30;
```

14.4.3 Re-run the query from above.

```
SELECT S.SS_SOLD_DATE_SK
     , R.SR_RETURNED_DATE_SK
     , S.SS_STORE_SK
     , S.SS_ITEM_SK
     , S.SS_SALES_PRICE
     , S.SS_SALES_PRICE
     , R.SR_RETURN_AMT
  FROM STORE_SALES  S
    INNER JOIN STORE_RETURNS  R
       on R.SR_ITEM_SK=S.SS_ITEM_SK
  WHERE  S.SS_ITEM_SK =4164;
```

The statement will timeout before it completes, and generate an error message.

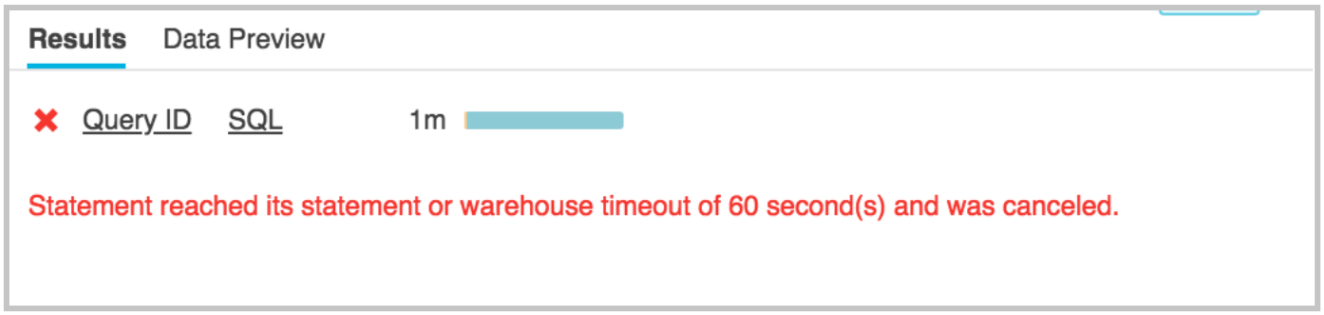

Figure 26: Error Message

14.4.4 Set the timeout back to its default.

```
ALTER WAREHOUSE [login]_query_wh UNSET STATEMENT_TIMEOUT_IN_SECONDS;
```

14.5 Use Query Tags

In this exercise, we will explore setting QUERY_TAG parameter to track queries executed within a session. This additional metadata can be queried later for tracking purposes: * by searching the INFORMATION_SCHEMA.QUERY_HISTORY table * or by viewing the Query Tag filter in the History tab in the UI.

14.5.1 See if an existing query tag is set.

```
SHOW PARAMETERS LIKE '%query_tag%' FOR SESSION;
```

14.5.2 Update the QUERY_TAG parameter:

```
ALTER SESSION SET QUERY_TAG = '[login]_join_test';

SHOW PARAMETERS LIKE '%query_tag%' FOR SESSION;
```

14.5.3 Run the JOIN explosion query from the previous exercise with a LIMIT.

```
SELECT s.ss_sold_date_sk
     , r.sr_returned_date_sk
     , s.ss_store_sk
     , r.sr_returned_date_sk
     , s.ss_item_sk
     , s.ss_sales_price
     , r.sr_return_amt
  FROM store_sales s
      INNER JOIN store_returns r
        ON  r.sr_item_sk=s.ss_item_sk
```



```
  WHERE s.ss_item_sk = 4164
  LIMIT 100;
```

14.5.4 Use the HISTORY UI and the Query Tag filter to find tagged queries.

It will look something like this:

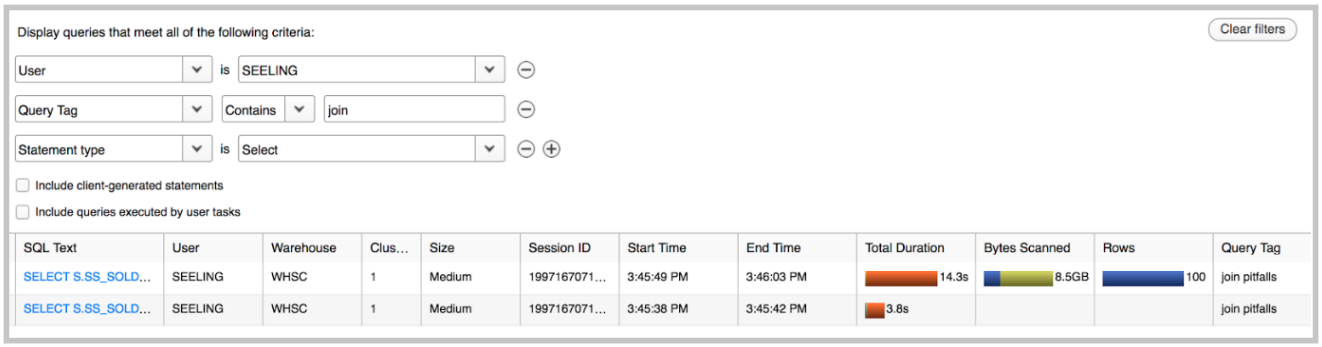

Figure 27: Show Parameters Results

14.5.5 Look for the query tag using the `INFORMATION_SCHEMA.QUERY_HISTORY` table function.

```
SELECT query_id, query_tag
  FROM TABLE (information_schema.query_history())
  WHERE query_tag LIKE '%[login]%';
```

14.5.6 Reset the query tag.

```
ALTER SESSION UNSET QUERY_TAG;

SHOW PARAMETERS LIKE '%query_tag%' FOR SESSION;
```


15 Unload Structured Data

Expect this lab to take approximately *40 minutes*.

15.1 Unload a Pipe-Delimited File to an Internal Stage

15.1.1 Open a worksheet and set your context:

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

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

```
CREATE OR REPLACE TABLE region AS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.REGION;
```

15.1.3 Unload the data to the REGION table stage.

Remember that a table stage is automatically created for each table. You will use MYPIPEFORMAT for the unload:

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

15.1.4 List the stage and verify the data is there:

```
LIST @%region;
```

15.1.5 (OPTIONAL) Download the files to your local system.

Use the `GET` command to download all files in the `REGION` table stage to local directory.

The Snowflake Web UI does not support the `GET` command. If you have the SnowSQL command line client installed, use it to connect to Snowflake and execute the `GET` command.

```
GET @%region file:///<path to dir> ; -- this is for Linux or MacOS
GET @%region file://c:<path to dir>; -- this is for Windows
```

After the files are downloaded to your local file system, open them with an editor and see what they contain.

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

```
REMOVE @%region;
```

15.2 Unload Part of a Table

15.2.1 Create a new table from SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS:

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

15.2.2 Unload the columns o_orderkey, o_orderstatus, and o_orderdate from your new table, into the table's stage:

Remember that a table stage is automatically created for every table. Use the default file format.

```
COPY INTO @%new_orders FROM
(SELECT o_orderkey, o_orderstatus, o_orderdate FROM new_orders);
```

15.2.3 Verify the output is in the stage:

```
LIST @%new_orders;
```

15.2.4 (OPTIONAL) Download the files to your local system.

Use the `GET` command to download all files in the `new_orders` table stage to local directory.

The Snowflake Web UI does not support the `GET` command. If you have the SnowSQL command line client installed, use it to connect to Snowflake and execute the `GET` command.

```
GET @%new_orders file:///<path> -- for Linux or MacOS
GET @%new_orders file://c:<path -- For Windows
```

How many files did you get? At what point did `COPY INTO` decide to split the files?

15.2.5 Remove the files from the stage:

```
REMOVE @%new_orders;
```


15.2.6 Repeat the unload with the selected columns, but this time specify SINGLE=TRUE in your COPY INTO command.

Also provide a name for the output file as part of the COPY INTO:

```
COPY INTO @%new_orders/new_orders.csv.gz FROM
(SELECT o_orderkey, o_orderstatus, o_orderdate FROM new_orders)
SINGLE=TRUE;
```

15.2.7 (OPTIONAL) Download the files to your local system.

Use the `GET` command to download all files in the `new_orders` table stage to local directory.

The Snowflake Web UI does not support the `GET` command. If you have the SnowSQL command line client installed, use it to connect to Snowflake and execute the `GET` command.

```
GET @%new_orders file:///<path> -- for Linux or MacOS
GET @%new_orders file://c:<path -- For Windows
```

15.2.8 Remove the file from the stage:

```
REMOVE @%new_orders;
```

15.3 JOIN and Unload a Table

15.3.1 Run a SELECT with a JOIN on the `REGION` and `NATION` tables.

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

15.3.2 Create a named internal stage.

```
CREATE STAGE [login]_stage;
```

15.3.3 Unload the JOINed data into the stage you created.

```
COPY INTO @[login]_stage FROM
(SELECT * FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."REGION" r JOIN
   "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."NATION" n
```



```
ON r.r_regionkey = n.n_regionkey);
```

15.3.4 Verify the file is in the stage.

```
LIST @[login]_stage;
```

15.3.5 (OPTIONAL) Download the files to your local system.

Use the `GET` command to download all files in the `@mystage` stage to local directory.

The Snowflake Web UI does not support the `GET` command. If you have the SnowSQL command line client installed, use it to connect to Snowflake and execute the `GET` command.

```
GET @mystage file:///<path> -- for Linux or MacOS
GET @mystage file://c:<path -- For Windows
```

15.3.6 Remove the file from the stage.

```
REMOVE @[login]_stage;
```

15.3.7 Remove the stage.

```
DROP STAGE [login]_stage;
```

16 Unload Semi-Structured Data

This lab will take approximately *30 minutes* to complete.

This lab will explore how to unload Semi-Structured Data (JSON and Parquet) from Snowflake tables into a Snowflake stage.

16.1 Unload Semi-Structured JSON Data

In this exercise, you will unload JSON data that has been loaded into the `TRAINING_DB.WEATHER.ISD_2019_TOTAL` table. This table contains data of global hourly weather observations compiled and recorded from over 35,000 stations worldwide during the year 2019. Upon completion, you will have unloaded the weather station data to a Snowflake stage separated by each country. In addition, we will explore how to implement a stored procedure to support a dynamic load path to unload the weather data by the country.

16.1.1 Semi Structured Data Types

First, explore the global weather data table.

```
use role training_role;
create warehouse if not exists [login]_wh;
use warehouse [login]_wh;
use schema TRAINING_DB.WEATHER;

describe table TRAINING_DB.WEATHER.ISD_2019_TOTAL;
```

The table contains two columns of type `VARIANT` and `TIMESTAMP_NTZ(9)`.

Row	name	type	kind	null?	default
1	V	VARIANT	COLUMN	Y	NULL
2	T	TIMESTAMP_NTZ(9)	COLUMN	Y	NULL

Figure 28: Describe Table

16.1.2 Query the VARIANT column

Using Snowflake dot notation, query the `v` column to view values in `TRAINING_DB.WEATHER.ISD_2019_TOTAL` for weather stations in the US (country = 'US').

```
select v from TRAINING_DB.WEATHER.ISD_2019_TOTAL
where v:station.country = 'US'
limit 10;
```

Click on a row to view the JSON stored in the VARIANT data type; it should look similar to:

```
{
  "data": {
    "observations": [
      {
        "air": {
          "dew-point": 999.9,
          "dew-point-quality-code": "9",
          "temp": 1.9,
          "temp-quality-code": "1"
        },
        "atmospheric": {
          "pressure": 10318,
          "pressure-quality-code": "1"
        },
        "dt": "2019-03-26T19:00:00",
        "sky": {
          "ceiling": 99999,
          "ceiling-quality-code": "9"
        },
        "visibility": {
          "distance": 999999,
          "distance-quality-code": "9"
```



```
        },
        "wind": {
          "direction-angle": 220,
          "direction-quality-code": "1",
          "speed-quality-code": "1",
          "speed-rate": 67
        }
      }
    ]
  },
  "station": {
    "USAF": "997802",
    "WBAN": 99999,
    "coord": {
      "lat": 44.05,
      "lon": -86.517
    },
    "country": "US",
    "elev": 186,
    "id": "99780299999",
    "name": "BIG SABLE POINT",
    "state": "MI"
  }
}
```


16.1.3 Create Unload Objects

Now that you have defined the query to return the JSON data, you will use the `COPY INTO` command to unload the data to a stage.

Before running the `COPY INTO` command, create a `STAGE` and a named `FILE FORMAT`.

```
use warehouse [login]_wh;
create database if not exists [login]_db;
use [login]_db.public;

create or replace stage [login]_unload;

create or replace file format jsonformat
  type = 'JSON' COMPRESSION = AUTO;
```

16.1.4 Use the COPY command to unload JSON data.

Use the `COPY` command to unload all the rows from the `TRAINING_DB.WEATHER.ISD_2019_TOTAL` table into one or more compressed JSON files in the `[login]_unload` stage, with a max file size of 10MB. Also, prefix the unloaded file(s) with `json/weather/2019/US/stations` to organize the files first by year, and then by country code, in the stage. This will take a few minutes.

```
copy into @[login]_unload/json/weather/2019/US/stations from
(
  select v from TRAINING_DB.WEATHER.ISD_2019_TOTAL
```



```
  where v:station.country::string = 'US'
  order by v:station.state
)
file_format = (format_name = 'jsonformat') max_file_size=1024;
```

16.1.5 View the list of unloaded files.

```
list @[login]_unload/json/weather/2019/US/;
```

Examine the output and verify that path and file size meet the criteria specified in the COPY command.

Row	name	size	md5
1	boa_unload/json/weather/2019/US/stations_0_0_0.json.gz	1053392	78acad1f20fd7f0b650fb8800114e7bd
2	boa_unload/json/weather/2019/US/stations_0_0_1.json.gz	1053824	1622dbae2d508922c98aa90ae2adafcd
3	boa_unload/json/weather/2019/US/stations_0_0_10.json.gz	1055504	527663592a57c20dacfd4179097626cd
4	boa_unload/json/weather/2019/US/stations_0_0_11.json.gz	1055088	9509d766dcf1a46ed6ad42ae61be361e
5	boa_unload/json/weather/2019/US/stations_0_0_12.json.gz	1052560	1b80e9b37e5fe205ecfa3b04203bac80
6	boa_unload/json/weather/2019/US/stations_0_0_13.json.gz	1055104	b01413e4d35d6d79b6482d3d50804c67
7	boa_unload/json/weather/2019/US/stations_0_0_14.json.gz	1056000	a45efb4335446cd86934c932b0c79677
8	boa_unload/json/weather/2019/US/stations_0_0_15.json.gz	1054464	c2c09b63f7934a5011dd26fbe6ddcbe4
9	boa_unload/json/weather/2019/US/stations_0_0_16.json.gz	1053888	1f5bdf866377fd305a984090a1c398bf
10	boa_unload/json/weather/2019/US/stations_0_0_17.json.gz	1055632	861cd3f7d2525ab05df5c135acf48e54

Figure 29: List Unloaded JSON Files

16.2 Unload Semi-Structured JSON Data Using a Dynamic Path

Now that the weather station data has been unloaded to a Snowflake stage for a single country, the next step is to create a stored procedure to dynamically set the COUNTRY_CODE in the COPY command.

16.2.1 Create a Stored Procedure.

```
CREATE OR REPLACE PROCEDURE UNLOAD_ISD_2019_TOTAL(COUNTRY_CODE STRING)
    RETURNS STRING
    LANGUAGE JAVASCRIPT
    EXECUTE AS CALLER
    AS
    $$

    var result="";

    try {
      var unload_query = "copy into @[login]_unload/json/weather/2019/" +
```



```
                          COUNTRY_CODE +
                         "/stations from" +
                         "( select v from TRAINING_DB.WEATHER.ISD_2019_TOTAL" +
                         "  where v:station.country = '" + COUNTRY_CODE + "'" +
                         ") file_format = (type = 'json')" +
                         "  max_file_size=1024" +
                         "  overwrite=true;";

      var unload_query_results = snowflake.execute({sqlText: unload_query});

      result = "Success: Unloaded " + COUNTRY_CODE + " stations.";
    }
    catch (err)  {
        result =  "Failed: Code: " + err.code + "\n  State: " + err.state;
        result += "\n  Message: " + err.message;
        result += "\nStack Trace:\n" + err.stackTraceTxt;
    }

    return result;
    $$;
```

Examine the body of the stored procedure and note that the `COPY INTO` statement is nearly the same as that last step. The only modification was to add the `overwrite=true` option.

Next, invoke the `UNLOAD_ISD_2019_TOTAL` procedure, passing in the COUNTRY_CODE for the US.

```
CALL UNLOAD_ISD_2019_TOTAL('US');

LIST @[login]_unload/json/weather/2019/US/;
```

Confirm the number of rows from the results of the `LIST` command.

A Snowflake stored procedure can include procedural logic (branching and looping), which straight SQL does not support. This can provide error handling, and dynamically create a SQL statement and execute it.

16.2.2 Get a list of country codes.

Run the following query to find the list of distinct country codes in the `TRAINING_DB.WEATHER.ISD_2019_TOTAL` table.

```
select DISTINCT(v:station.country::string) from TRAINING_DB.WEATHER.ISD_2019_TOTAL;
```

There are hundreds of country codes in the data. On your own time, try to rewrite the stored procedure to run the provided query to get a LIST OF COUNTRY CODES, then use a loop to iterate through the list, running a `COPY INTO` statement for each `COUNTRY CODE`.

16.3 Unload Structured Data to a JSON File

In this exercise, you will unload the data in the relational table `SNOWSTORE.DWH.DIMCUSTOMERS` into a JSON file that meets the specified JSON syntax for storing and exchanging data with a third party vendor.

Below is a JSON document that meets the required specification:

```
/*
{
  "customer_info": {
    "key": 118,
    "id": 386,
    "city": "Gold Coast",
    "state": "Queensland"
    "country": "Australia"
  },
  "customer_pii": {
    "firstname": "Refugio ",
    "lastname": "Whittaker "
    "gender": "M",
    "latlong": [ 0, 0]
  }
}
*/
```

You will use the `VARIANT`, `OBJECT` and `ARRAY` data types to transform and unload the structured data source into the JSON document that meets the required specification.

16.3.1 Query the structured table.

Explore the columns and values of the `SNOWSTORE.DWH.DIMCUSTOMERS` table to determine what structuced columns will need to be mapped to the JSON document specification.

```
select * from SNOWSTORE.DWH.DIMCUSTOMERS;

describe table SNOWSTORE.DWH.DIMCUSTOMERS;
```

Looking at the columns names, we can apply the following mapping rules:

```
/*
| JSON key name | Table Column Name |
|---------------|-------------------|
| key           | CUSTOMERKEY       |
| id            | CUSTOMERID        |
| city          | CITY              |
| state         | STATE             |
| country       | COUNTRY           |
| firstname     | FIRSTNAME         |
| lastname      | LASTNAME          |
| gender        | GENDER            |
| latlong       | NO MATCH          |
*/
```

16.3.2 Create the JSON objects.

Use the semi-structured data function `OBJECT_CONSTRUCT` to create the objects (key:value pairs) with the corresponding values in `SNOWSTORE.DWH.DIMCUSTOMERS` table.


```
select object_construct('key', CUSTOMERKEY,
                        'id', CUSTOMERID,
                        'city', CITY,
                        'state', STATE,
                        'country', COUNTRY,
                        'firstname', FIRSTNAME,
                        'lastname', LASTNAME,
                        'gender', GENDER)
from SNOWSTORE.DWH.DIMCUSTOMERS;
```

Examine the output, we are getting close. The output looks something like that shown below. Note, we do not need to be concerned about ordering of the keys only that the keys match the specification.

```
/*
{
  "city": "Gold Coast",
  "country": "Australia",
  "firstname": "Refugio ",
  "gender": "Male",
  "id": 386,
  "key": 118,
  "lastname": "Whittaker ",
  "state": "Queensland"
}
*/
```

16.3.3 Create nested JSON objects.

Next, create and add the objects (key:value pairs) to the `customer_info` and `customer_pii` objects.

```
select object_construct(
          'customer_info',object_construct(
                            'key',CUSTOMERKEY,
                            'id',CUSTOMERID,
                            'city',CITY,
                            'state',STATE,
                            'country',COUNTRY),
          'customer_pii',object_construct(
                            'firstname',FIRSTNAME,
                            'lastname',LASTNAME,
                            'gender',GENDER)
)
from SNOWSTORE.DWH.DIMCUSTOMERS;
```

Examine the output. All that is left is to create the `latlong` array.

```
/*
{
  "customer_info": {
    "city": "Gold Coast",
    "country": "Australia",
    "id": 386,
    "key": 118,
```



```
    "state": "Queensland"
  },
  "customer_pii": {
    "firstname": "Refugio ",
    "gender": "Male",
    "lastname": "Whittaker  "
  }
}
*/
```

16.3.4 Add the `latlong` array.

Create the `latlong` object and use the `array_construct` method with values of 0.0, since the structured data does not contain that information.

```
select object_construct(
          'customer_info',object_construct(
                              'key',CUSTOMERKEY,
                              'id',CUSTOMERID,
                              'city',CITY,
                              'state',STATE,
                              'country',COUNTRY),
          'customer_pii',object_construct(
                              'firstname',FIRSTNAME,
                              'lastname',LASTNAME,
                              'gender',GENDER,
                              'latlong',array_construct(0.0,0.0))
)
from SNOWSTORE.DWH.DIMCUSTOMERS;
```

Examine the output. Notice that there is some unwanted whitespace in some of the string values and that the `gender` value should be a single upper case letter.

```
/*
{
  "customer_info": {
    "city": "Gold Coast",
    "country": "Australia",
    "id": 386,
    "key": 118,
    "state": "Queensland"
  },
  "customer_pii": {
    "LATLONG": [
      0,
      0
    ],
    "firstname": "Refugio ",
    "gender": "Male",
    "lastname": "Whittaker "
  }
}
*/
```


16.3.5 Clean up the final output.

Use the TRIM and REPLACE functions to remove the unwanted whitespace for the columns that have VARCHAR types and replace the gender value with the correct abbreviation.

```
select object_construct(
          'customer_info',object_construct(
                              'key',CUSTOMERKEY,
                              'id',CUSTOMERID,
                              'city',TRIM(CITY),
                              'state',TRIM(STATE),
                              'country',TRIM(COUNTRY)),
          'customer_pii',object_construct(
                              'firstname',TRIM(FIRSTNAME),
                              'lastname',TRIM(LASTNAME),
                              'gender',case
                                 when UPPER(TRIM(GENDER)) = 'MALE' then 'M'
                                 when UPPER(TRIM(GENDER)) = 'FEMALE' then 'F'
                                 else 'O' end,
                              'LATLONG',array_construct(0.0,0.0))
)
from SNOWSTORE.DWH.DIMCUSTOMERS;
```

16.3.6 Unload the JSON data.

Now that the select statement is complete, add it to the COPY INTO command:

```
copy into @[login]_unload/json/customers from
(
  select object_construct(
          'customer_info',object_construct(
                              'key',TRIM(CUSTOMERKEY),
                              'id',TRIM(CUSTOMERID),
                              'city',TRIM(CITY),
                              'state',TRIM(STATE),
                              'country',TRIM(COUNTRY)),
          'customer_pii',object_construct(
                              'firstname',TRIM(FIRSTNAME),
                              'lastname',TRIM(LASTNAME),
                              'gender',case
                                 when UPPER(TRIM(GENDER)) = 'MALE' then 'M'
                                 when UPPER(TRIM(GENDER)) = 'FEMALE' then 'F'
                                 else 'O' end,
                              'LATLONG',array_construct(0.0,0.0))
  )
  from SNOWSTORE.DWH.DIMCUSTOMERS
)
file_format = (format_name = 'jsonformat');
```

16.3.7 List the file(s) generated.


```
list @[login]_unload/json/customers;
```

16.3.8 Query the JSON file from the stage.

```
select metadata$filename, metadata$file_row_number, $1
from @[login]_unload/json/customers_0_0_0.json.gz
(file_format => jsonformat);
```

16.4 Unload Semi-Structured Parquet Data

In this exercise, you will unload JSON data in the `TRAINING_DB.WEATHER.ISD_DAILY` table in Parquet format, to a Snowflake named stage organized by each country.

This table contains global hourly weather observation data compiled and recorded from over 35,000 stations worldwide during the year from 1901 to 2019. Since Parquet is a column-oriented format, you will transform the semi-structured JSON data into a structured format.

Upon completion, you will have unloaded the weather station data to a Snowflake stage for a single country for the year 2019.

16.4.1 Create Unload Objects.

Before creating the `COPY INTO` command, create a named `FILE FORMAT` object.

```
create or replace file format parquetformat
  type = 'PARQUET' COMPRESSION = AUTO;
```

16.4.2 Query a VARIANT column.

```
select v from TRAINING_DB.WEATHER.ISD_2019_DAILY limit 25;
```

Examine the results. Notice how the `observations` array contains station measurements for each hour of the day.

16.4.3 FLATTEN the output.

Use the FLATTEN function, a LATERAL join, and dot notation to create a row for each station's observations array.

```
 SELECT weather.t as date,
       v:station.name::STRING AS station,
       v:station.country::STRING AS country,
       v:station.id::STRING AS id,
       observations.value:dt::timestamp AS time,
```



```
       observations.value:air.temp::FLOAT AS temp_celsius,
       observations.value:air."temp-quality-code"::STRING AS temp_qc_code,
       observations.value:air."dew-point"::FLOAT AS dew_point,
       observations.value:air."dew-point-quality-code"::STRING AS
          dew_point_qc_code,
       observations.value:atmospheric.pressure::FLOAT AS atm_pressure,
       observations.value:atmospheric."pressure-quality-code"::STRING AS
          atm_pressure_qc_code,
       observations.value:sky.ceiling::FLOAT AS sky_ceiling,
       observations.value:sky."ceiling-quality-code"::STRING AS
          sky_ceiling_qc_code,
       observations.value:visibility.distance::FLOAT AS vis_distance,
       observations.value:visibility."distance-quality-code"::STRING AS
          vis_distance_qc_code,
       observations.value:wind."direction-angle"::FLOAT AS wind_direction,
       observations.value:wind."direction-quality-code"::STRING AS
          wind_direction_qc_code,
       observations.value:wind."speed-quality-code"::STRING AS wind_speed_qc_code,
       observations.value:wind."speed-rate"::FLOAT AS wind_speed
FROM TRAINING_DB.WEATHER.isd_daily weather,
LATERAL FLATTEN(input => v:data.observations) observations
WHERE date BETWEEN to_timestamp_ntz('2019-01-01 00:00:00') AND
   to_timestamp_ntz('2019-12-31 23:59:59')
AND country = 'US'
LIMIT 500;
```

Examine the results. There is one row for each measurement in each station.

Row	DATE	STATION	COUNTRY	ID	TIME	TEMP_CELSIUS.	TEMP_QC_CODE
1	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 00:53:00.000	40	5
2	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 01:53:00.000	37.8	5
3	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 02:53:00.000	36.7	5
4	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 03:53:00.000	36.7	5
5	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 04:53:00.000	35	5
6	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 05:53:00.000	34.4	5
7	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 06:53:00.000	32.2	5
8	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 07:53:00.000	31.1	5
9	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 07:59:00.000	999.9	9
10	2019-09-06	IMPERIAL COUNTY AIRPORT	US	74718503144	2019-09-06 08:53:00.000	29.4	5

Figure 30: Query Results

16.4.4 Unload the structured data to a Parquet file.

Use `COPY INTO` to unload the transformed JSON data to Parquet. To retain the column names in the output file, use the `HEADER = TRUE` option.

```
copy into @[login]_unload/parquet/weather/2019/US/stations from
(
  SELECT weather.t as date,
```



```
       v:station.name::STRING AS station,
       v:station.country::STRING AS country,
       v:station.id::STRING AS id,
       observations.value:dt::timestamp AS time,
       observations.value:air.temp::FLOAT AS temp_celsius,
       observations.value:air."temp-quality-code"::STRING AS temp_qc_code,
       observations.value:air."dew-point"::FLOAT AS dew_point,
       observations.value:air."dew-point-quality-code"::STRING AS
          dew_point_qc_code,
       observations.value:atmospheric.pressure::FLOAT AS atm_pressure,
       observations.value:atmospheric."pressure-quality-code"::STRING AS
          atm_pressure_qc_code,
       observations.value:sky.ceiling::FLOAT AS sky_ceiling,
       observations.value:sky."ceiling-quality-code"::STRING AS
          sky_ceiling_qc_code,
       observations.value:visibility.distance::FLOAT AS vis_distance,
       observations.value:visibility."distance-quality-code"::STRING AS
          vis_distance_qc_code,
       observations.value:wind."direction-angle"::FLOAT AS wind_direction,
       observations.value:wind."direction-quality-code"::STRING AS
          wind_direction_qc_code,
       observations.value:wind."speed-quality-code"::STRING AS wind_speed_qc_code,
       observations.value:wind."speed-rate"::FLOAT AS wind_speed
FROM TRAINING_DB.WEATHER.isd_daily weather,
LATERAL FLATTEN(input => v:data.observations) observations
WHERE date BETWEEN to_timestamp_ntz('2019-01-01 00:00:00') AND
   to_timestamp_ntz('2019-12-31 23:59:59')
AND country = 'US'

)
file_format = (format_name = 'parquetformat')
OVERWRITE = TRUE
HEADER = TRUE;
```

16.4.5 List the files in the stage.

```
list @[login]_unload/parquet/weather/2019/US/stations;
```

16.4.6 Query the PARQUET file in the stage.

```
select $1 from
   @[login]_unload/parquet/weather/2019/US/stations_0_0_0.snappy.parquet
   (file_format => parquetformat);
```

17 Data Sharing

This lab will take approximately *30 minutes* to complete.

Secure Data Sharing enables account-to-account sharing of data through Snowflake database tables, secure views, and secure UDFs.

In this exercise you will learn how to setup two Snowflake accounts for data sharing. The first account will be the data **[provider-account]** . The second account will be the data **[consumer-account]**. You will then perform the steps to enable sharing selected objects in a database in the **[provider-account]** with the **[consumer-account]**. No actual data is copied or transferred between accounts. All data sharing is accomplished through Snowflake's unique services layer and metadata store.

17.1 Setup Browsers for Data Sharing

17.1.1 Open two browser windows side-by-side.

17.1.2 In browser 1, enter the Snowflake **[provider-account]** URL, login, navigate to worksheets and load the script into a new worksheet.

In this worksheet, you will only execute commands that are surrounded by PROVIDER comments:

```
-- PROVIDER --
```

17.1.3 Rename the worksheet to something that contains the word **provider**.

17.1.4 In browser 2, enter the Snowflake **[consumer-account]** URL, login, navigate to worksheets and load the script into a new worksheet.

In this worksheet, you will only execute commands that are surrounded by CONSUMER comments:

```
-- CONSUMER --
```


17.1.5 Rename the worksheet to something that contains the word **consumer**.

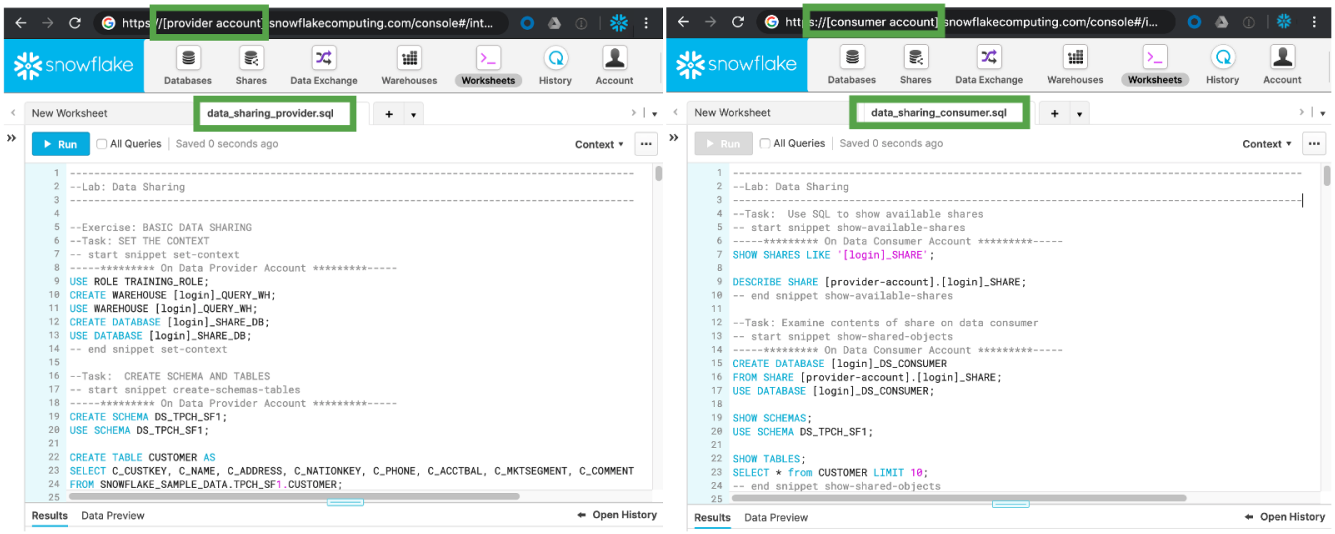

Figure 31: Data Sharing Browser Setup

17.2 Basic Data Sharing

In this exercise you will create a basic data share and share data.

Perform the following steps in the **[provider-account]** in browser 1.

17.2.1 Set the context

```
-- PROVIDER --
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_QUERY_WH;
USE WAREHOUSE [login]_QUERY_WH;
CREATE DATABASE [login]_SHARE_DB;
USE DATABASE [login]_SHARE_DB;
-- PROVIDER --
```

17.2.2 Create schema and tables

```
-- PROVIDER --
CREATE SCHEMA DS_TPCH_SF1;
USE SCHEMA DS_TPCH_SF1;

CREATE TABLE CUSTOMER AS
SELECT C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE, C_ACCTBAL,
    C_MKTSEGMENT, C_COMMENT
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER;

CREATE TABLE ORDERS AS
```



```
SELECT O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE,
       O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS;
-- PROVIDER --
```

17.2.3 Create empty share

An empty share is a shell that you can later use to share actual objects.

```
-- PROVIDER --
CREATE SHARE [login]_SHARE;
-- PROVIDER --
```

17.2.4 Grant object privileges to the share

```
-- PROVIDER --
GRANT USAGE ON DATABASE [login]_SHARE_DB TO SHARE [login]_SHARE;

GRANT USAGE ON SCHEMA [login]_SHARE_DB.DS_TPCH_SF1 TO SHARE [login]_SHARE;

GRANT SELECT ON TABLE [login]_SHARE_DB.DS_TPCH_SF1.CUSTOMER TO SHARE [login]_SHARE;

GRANT SELECT ON TABLE [login]_SHARE_DB.DS_TPCH_SF1.ORDERS TO SHARE [login]_SHARE;
-- PROVIDER --
```

17.2.5 Add account to the share

```
-- PROVIDER --
ALTER SHARE [login]_SHARE SET ACCOUNTS=[consumer-account];
-- PROVIDER --
```

17.2.6 Validate the share configuration

```
-- PROVIDER --
SHOW GRANTS TO SHARE [login]_SHARE;
-- PROVIDER --
```

17.3 Create database on data consumer account from tables shared on the data provider server

Perform all the steps in this section on **[consumer-account]** in browser 2.

17.3.1 View the in inbound shares in the web ui.

17.3.2 Click the “Shares” icon.

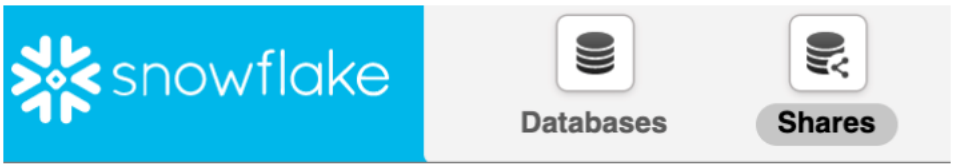

Figure 32: Shares Icon

17.3.3 You should see something like:

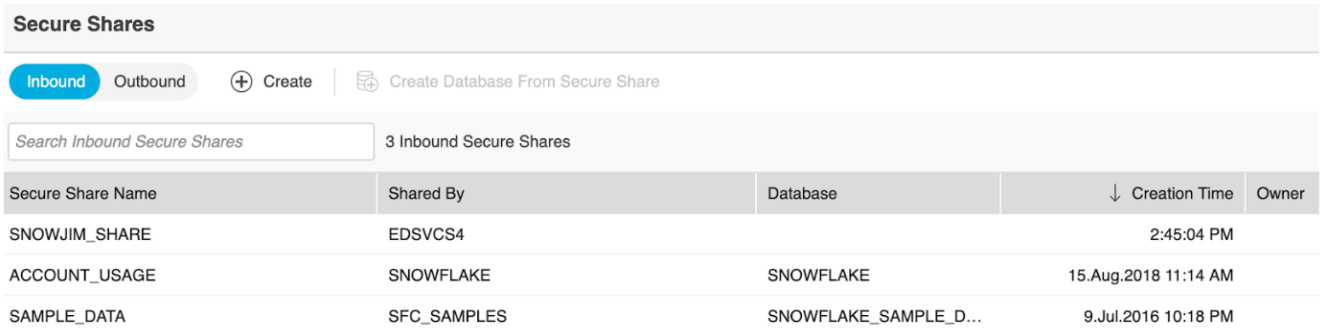

Figure 33: Inbound Shares

17.3.4 View Outbound Shares on the Data Provider

17.3.5 On the data provider server, click on the “Shares” button and then hit the Outbound button. You should see the share that you setup.

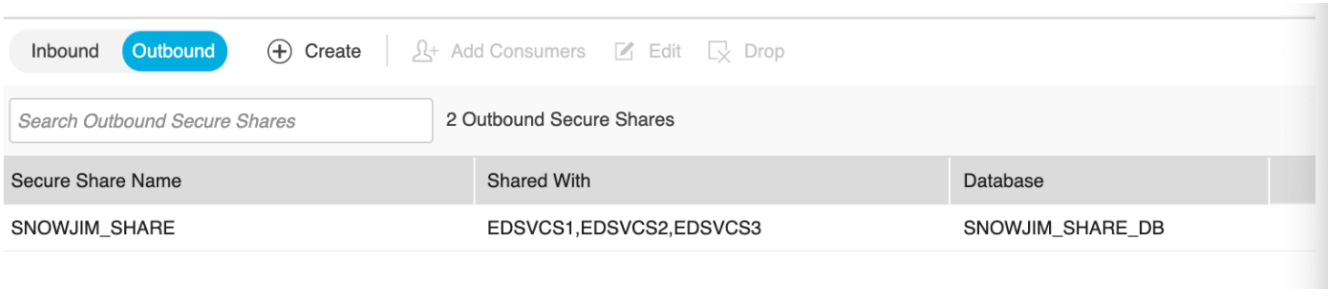

Figure 34: Outbound Shares

17.3.6 Create a Warehouse


```
-- CONSUMER --
USE ROLE TRAINING_ROLE;

CREATE OR REPLACE WAREHOUSE [login]_QUERY_WH
   WAREHOUSE_SIZE = 'LARGE'
   AUTO_SUSPEND = 300
   AUTO_RESUME = TRUE
   MIN_CLUSTER_COUNT = 1
   MAX_CLUSTER_COUNT = 1
   SCALING_POLICY = 'STANDARD'
   COMMENT = 'Training WH for completing hands on lab queries';
-- CONSUMER --
```

17.3.7 Use SQL to show available shares

```
-- CONSUMER --
SHOW SHARES LIKE '[login]_SHARE';

DESCRIBE SHARE [provider-account].[login]_SHARE;
-- CONSUMER --
```

17.3.8 Examine contents of share on data consumer

```
-- CONSUMER --
CREATE DATABASE [login]_DS_CONSUMER
FROM SHARE [provider-account].[login]_SHARE;
USE DATABASE [login]_DS_CONSUMER;

SHOW SCHEMAS;
USE SCHEMA DS_TPCH_SF1;

SHOW TABLES;
SELECT * from CUSTOMER LIMIT 10;
-- CONSUMER --
```

17.3.9 Use Public Role, and test ability to query share

```
-- CONSUMER --
GRANT USAGE ON WAREHOUSE [login]_query_wh TO ROLE public;

USE ROLE PUBLIC;
USE DATABASE [login]_DS_CONSUMER;
--This command will fail because PUBLIC does not have access to the database

USE ROLE TRAINING_ROLE;
GRANT USAGE ON DATABASE [login]_DS_CONSUMER TO ROLE public;
--You should have received an error message: you cannot add privileges to a share

GRANT IMPORTED PRIVILEGES ON DATABASE [login]_DS_CONSUMER TO ROLE PUBLIC;
```



```
USE ROLE PUBLIC;
USE DATABASE [login]_DS_CONSUMER;

SHOW SCHEMAS;
USE SCHEMA DS_TPCH_SF1;

SELECT * from CUSTOMER LIMIT 10;
-- CONSUMER --
```

Note: You will run into a few error messages as you run each statement above. Keep going and the subsequent queries will make the necessary changes.

17.4 Create A Secure View And Add It To The Share

Perform all the steps in this section on **[provider-account]** in browser 1.

17.4.1 Create a schema

```
-- PROVIDER --
USE DATABASE [login]_share_db;
CREATE SCHEMA private;
-- PROVIDER --
```

17.4.2 Create a mapping table

A "mapping table" is only required if you wish to share the data in the base table with multiple consumer accounts and share specific rows in the table with specific accounts.

```
-- PROVIDER --
CREATE OR REPLACE TABLE PRIVATE.SHARING_ACCESS(
  C_CUSTKEY STRING,
  SNOWFLAKE_ACCOUNT STRING);
-- PROVIDER --
```

17.4.3 Populate mapping table

```
-- PROVIDER --
INSERT INTO PRIVATE.SHARING_ACCESS (C_CUSTKEY, SNOWFLAKE_ACCOUNT)
SELECT C_CUSTKEY,
       CASE WHEN C_CUSTKEY BETWEEN 1 AND 20 THEN '[consumer-account]'
            ELSE 'UNKNOWN'
            END AS SNOWFLAKE_ACCOUNT
FROM DS_TPCH_SF1.CUSTOMER
WHERE C_CUSTKEY BETWEEN 1 AND 50;
-- PROVIDER --
```


17.4.4 Create a secure view

Remember a secure view hides the SQL used to create the view and runs the optimizer after the secured data is filtered out meaning that the query will not return an access error message. Unauthorized values will appear as not in the table

```
-- PROVIDER --
CREATE OR REPLACE SECURE VIEW DS_TPCH_SF1.CUST_SENSITIVE_DATA_VW AS
SELECT SD.C_CUSTKEY,
    SD.C_NAME,
    SD.C_ADDRESS,
    SD.C_NATIONKEY,
    SD.C_PHONE
FROM DS_TPCH_SF1.CUSTOMER SD
INNER JOIN PRIVATE.SHARING_ACCESS SA
ON SD.C_CUSTKEY = SA.C_CUSTKEY
AND UPPER(SA.SNOWFLAKE_ACCOUNT) = UPPER(CURRENT_ACCOUNT());
-- PROVIDER --
```

17.4.5 Validate the table and secure view

```
-- PROVIDER --
SELECT *
FROM DS_TPCH_SF1.CUST_SENSITIVE_DATA_VW;
-- should return 0 rows because the provider account is not mapped
-- PROVIDER --
```

17.4.6 Validate the secure view by simulating data consumer

```
-- PROVIDER --
ALTER SESSION SET SIMULATED_DATA_SHARING_CONSUMER='[consumer-account]';
SELECT *
FROM DS_TPCH_SF1.CUST_SENSITIVE_DATA_VW;
-- This should return 20 rows because simulated consumer account is mapped!
ALTER SESSION UNSET SIMULATED_DATA_SHARING_CONSUMER;
-- PROVIDER --
```

17.4.7 Add the secure view to the share

```
-- PROVIDER --
GRANT SELECT ON [login]_SHARE_DB.DS_TPCH_SF1.CUST_SENSITIVE_DATA_VW TO SHARE
   [login]_SHARE;
```

17.4.8 Confirm grants on the share


```
-- PROVIDER --
SHOW GRANTS TO SHARE [login]_SHARE;
-- PROVIDER --
```

17.5 Data Consumer - Use A Shared Database

Perform all the steps in this section on **[consumer-account]** in browser 2.

17.5.1 Set context and create warehouse if it does not exist

```
-- CONSUMER --
USE ROLE TRAINING_ROLE;
CREATE OR REPLACE WAREHOUSE [login]_LOAD_WH
   WAREHOUSE_SIZE = 'SMALL'
   AUTO_SUSPEND = 300
   AUTO_RESUME = TRUE
   MIN_CLUSTER_COUNT = 1
   MAX_CLUSTER_COUNT = 1
   SCALING_POLICY = 'STANDARD'
   COMMENT = 'Training WH for completing hands on lab queries';
-- CONSUMER --
```

17.5.2 View available shares

```
-- CONSUMER --
SHOW SHARES LIKE '[login]_SHARE';
DESCRIBE SHARE [provider-account].[login]_SHARE;
-- CONSUMER --
```

17.5.3 Create a database from the share and grant privileges

```
-- CONSUMER --
CREATE DATABASE [login]_DS_CONSUMER
FROM SHARE [provider-account].[login]_SHARE;
--this will generate an error, because the database already exists, from the
--first time the consumer ingested the share.  The provider did not create a
--new share, but added objects to an existing share.
GRANT IMPORTED PRIVILEGES ON DATABASE [login]_DS_CONSUMER TO ROLE TRAINING_ROLE;
-- CONSUMER --
```

17.5.4 Validate shared objects

```
-- CONSUMER --
SHOW DATABASES LIKE '[login]_DS_CONSUMER';
SHOW SCHEMAS IN DATABASE [login]_DS_CONSUMER;
```



```
SHOW TABLES IN SCHEMA [login]_DS_CONSUMER.DS_TPCH_SF1;
SHOW VIEWS IN SCHEMA [login]_DS_CONSUMER.DS_TPCH_SF1;
```

17.5.5 Validate shared objects by running queries

```
-- CONSUMER --
SELECT COUNT(*)
FROM [login]_DS_CONSUMER.DS_TPCH_SF1.CUST_SENSITIVE_DATA_VW;
-- expected row count =  20
SELECT COUNT(*)
FROM [login]_DS_CONSUMER.DS_TPCH_SF1.CUSTOMER;
-- expected row count = 150000
SELECT COUNT(*)
FROM [login]_DS_CONSUMER.DS_TPCH_SF1.ORDERS;
-- expected row count = 1500000
-- CONSUMER --
```

17.6 Remove objects

Perform the next steps in the **[provider-account]** in browser 1.

17.6.1 Remove the share

```
-- PROVIDER --
USE [login]_SHARE_DB;
DESCRIBE SHARE [login]_SHARE;
-- Revoke access to the ORDERS table:
REVOKE SELECT ON TABLE [login]_SHARE_DB.DS_TPCH_SF1.ORDERS FROM SHARE
   [login]_SHARE;
-- PROVIDER --
```

17.6.2 Confirm that the table was removed

```
-- PROVIDER --
DESCRIBE SHARE [login]_SHARE;
-- PROVIDER --
```

17.6.3 Confirm that the ORDERS table was revoked

Perform the next steps in the **[consumer-account]** in browser 2.

```
-- CONSUMER --
DESC SHARE [provider-account].[login]_SHARE;
SELECT COUNT (*) FROM [login]_DS_CONSUMER.DS_TPCH_SF1.ORDERS;
-- should fail with SQL compilation error
-- CONSUMER --
```


17.6.4 Verify table data

```
-- CONSUMER --
SELECT MIN(C_CUSTKEY)
  FROM [login]_DS_CONSUMER.DS_TPCH_SF1.CUSTOMER;
-- expected result = 1
-- CONSUMER --
```

17.6.5 Delete a row in the CUSTOMER table shared with the consumer account.

Perform this step on the **[provider-account]** in browser 1.

```
-- PROVIDER --
DELETE
  FROM [login]_SHARE_DB.DS_TPCH_SF1.CUSTOMER
  WHERE C_CUSTKEY = 1;
-- expected result = 1 Rows deleted
-- PROVIDER --
```

17.6.6 Verify the row was deleted

Perform this step on the **[consumer-account]** in browser 2.

```
-- CONSUMER --
SELECT MIN(C_CUSTKEY)
  FROM [login]_DS_CONSUMER.DS_TPCH_SF1.CUSTOMER;
-- expected result = 2. This is different from before because the row
-- with C_CUSTKEY=1 was removed by the provider.
-- CONSUMER --
```

17.7 The Power Of Secure User-defined Functions For Protecting Shared Data

Secure Views And Their Limitations

Today, most data sharing in Snowflake uses secure views. Secure views are a great way for a data owner to grant other Snowflake users secure access to select subsets of their data.

Secure views are effective for enforcing cell-level security in multi-tenant situations. This includes software-as-a-service (SaaS) providers granting access to each of their customers, while allowing each customer to see only their specific rows of data from each table. However, there is nothing preventing another user from running a SELECT * query against the secure view and then exporting all the data that's visible to them.

In many situations, allowing a data consumer to see and export the raw data is completely acceptable. However, in other situations, such as when monetizing data, the most valuable analyses are often run against low-level and raw data, and allowing a data consumer to export the raw data is not desirable. Furthermore, when PII and PHI are involved, privacy policies and government regulations often do not permit providing data access to other parties.

Perform the next steps on the **[provider-account]** in browser 1.

17.7.1 The Power Of Secure UDFs

Secure UDFs are small pieces of SQL or JavaScript code that securely operate against raw data, but provide only a constrained set of outputs in response to specific inputs. For example, imagine a retailer that wants to allow its suppliers to see which items from other suppliers are commonly sold together with theirs. This is known as market basket analysis.

Using the TCP-DS sample data set that's available to all users from the Shares tab within Snowflake, we can run the following SQL commands to create a test data set and perform a market basket analysis:

```
-- PROVIDER --
CREATE DATABASE IF NOT EXISTS [login]_UDF_DEMO;
CREATE SCHEMA IF NOT EXISTS [login]_UDF_DEMO.PUBLIC;

CREATE OR REPLACE TABLE [login]_udf_demo.public.sales AS
(SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE_SALES
 SAMPLE BLOCK (1));

select 6139 as input_item
     , ss_item_sk as basket_Item
     , count(distinct ss_ticket_number) baskets
  from [login]_udf_demo.public.sales
  where ss_ticket_number in
         (select ss_ticket_number
             from [login]_udf_demo.public.sales
             where ss_item_sk = 6139)
  group by ss_item_sk
  order by 3 desc, 2;
-- PROVIDER --
```

17.7.2 Create a Secure UDF

This example returns the items that sold together with item #6139. This example outputs only aggregated data, which is the number of times various other products are sold together, in the same transaction, with item #6139. This SQL statement needs to operate across all of the raw data to find the right subset of transactions. To enable this type of analysis while preventing the user who is performing the analysis from seeing the raw data, we wrap this SQL statement in a secure UDF and add an input parameter to specify the item number we are selecting for market basket analysis, as follows:

```
-- PROVIDER --
CREATE OR REPLACE SECURE FUNCTION
    [login]_UDF_DEMO.PUBLIC.GET_MARKET_BASKET(INPUT_ITEM_SK NUMBER(38))
RETURNS TABLE (INPUT_ITEM NUMBER(38,0), BASKET_ITEM_SK NUMBER(38,0),NUM_BASKETS
    NUMBER(38,0))
AS
 'SELECT INPUT_ITEM_SK
       , SS_ITEM_SK BASKET_ITEM
       , COUNT(DISTINCT SS_TICKET_NUMBER) BASKETS
```



```
    FROM [login]_UDF_DEMO.PUBLIC.SALES
    WHERE SS_TICKET_NUMBER IN (SELECT SS_TICKET_NUMBER FROM
       [login]_UDF_DEMO.PUBLIC.SALES WHERE SS_ITEM_SK = INPUT_ITEM_SK)
GROUP BY SS_ITEM_SK
ORDER BY 3 DESC, 2';

SELECT * FROM TABLE([login]_UDF_DEMO.PUBLIC.GET_MARKET_BASKET(6139));
-- PROVIDER --
```

We can then call this function and specify any item number as an input, and we will get the same results we received when running the SQL statement directly. Now, we can grant a specified user access to this function while preventing the user from accessing the underlying transactional data.

17.7.3 How To Share Secure UDFs

To share a secure UDF, we can then grant usage rights on the secure UDF to a Snowflake share. This gives other specified Snowflake accounts the ability to run the secure UDF, but does not grant any access rights to the data in the underlying tables.

```
-- PROVIDER --
USE DATABASE [login]_udf_demo;
CREATE SHARE IF NOT EXISTS [login]_UDF_DEMO_SHARE;
GRANT USAGE ON DATABASE [login]_UDF_DEMO TO SHARE [login]_UDF_DEMO_SHARE;
GRANT USAGE ON SCHEMA [login]_UDF_DEMO.PUBLIC to share [login]_UDF_DEMO_SHARE;
GRANT USAGE ON FUNCTION [login]_UDF_DEMO.PUBLIC.get_market_basket(number) to share
   [login]_UDF_DEMO_SHARE;
ALTER SHARE [login]_UDF_DEMO_SHARE ADD ACCOUNTS=[consumer-account];
-- PROVIDER --
```

17.7.4 Create database from share and run query on the data consumer.

Perform this step on the **[consumer-account]** in browser 2.

We can run the secure UDF from the share using the second account's virtual warehouse. However, from the second account, we cannot select any data from the underlying tables, determine anything about the names or structures of the underlying tables, or see the code behind the secure UDF.

```
-- CONSUMER --
USE ROLE TRAINING_ROLE;
CREATE DATABASE [login]_UDF_TEST FROM SHARE
   [provider-account].[login]_UDF_DEMO_SHARE;
GRANT IMPORTED PRIVILEGES ON DATABASE [login]_UDF_TEST TO ROLE PUBLIC;
USE DATABASE [login]_UDF_TEST;
SELECT * FROM TABLE([login]_UDF_TEST.PUBLIC.GET_MARKET_BASKET(6139));
-- CONSUMER --
```

17.7.5 Examine Share from the data provider.

Perform this step on the **[provider-account]** in browser 1.


```
-- PROVIDER --
DESCRIBE SHARE [login]_UDF_DEMO_SHARE;
-- PROVIDER --
```

The secure UDF is essentially using the data access rights of its creator, but allowing itself to be run by another Snowflake account that has access rights to run it. With Snowflake Data Sharing, the compute processing for secure UDFs runs in the context of, and is paid for by, the data consumer using the consumer's virtual warehouse, against the function provider's single encrypted copy of the underlying data.

This ability to share a secure UDF enables a myriad of secure data sharing and data monetization use cases, including the ability to share raw and aggregated data and powerful analytical functions, while also protecting the secure UDF's code. It also prevents other parties from directly viewing or exporting the underlying encrypted data.

18 Use Dynamic Data Masking

This lab will take approximately *20 minutes* to complete.

A membership program needs to develop an analytic application using data which contains Personally Identifiable Information (PII). The production environment will need to provide the necessary restricted access. In order to develop the application, the development team requires a development environment to build, code and test the application against the PII data.

In this lab, you will create a development environment that securely provides the required production data to the development team using Snowflake Dynamic Data Masking.

18.1 Identify PII Data

The table `TRAINING_DB.TRAININGLAB.MEMBERS`, which you will use for this lab, contains PII data.

18.1.1 Query the table.

```
USE ROLE training_role;
USE WAREHOUSE [login]_wh;
USE DATABASE training_db;
USE SCHEMA traininglab;

DESCRIBE TABLE members;

SELECT firstname, lastname, age, email FROM members LIMIT 10;
```

By examining the columns names and their values it is clear that the `MEMBERS` table contains multiple columns of PII data.

18.1.2 Create a new database using CLONE.

To prepare the development environment, create a new schema and clone the MEMBERS table to it.

```
CREATE OR REPLACE SCHEMA [login]_db.mask_lab;
USE SCHEMA [login]_db.mask_lab;

CREATE OR REPLACE TABLE members CLONE training_db.traininglab.members;
```

This created a MEMBERS table in the new schema, but changed the ownership of the table to TRAINING_ROLE (the role that created it).

18.2 Create Masking Policies

18.2.1 Create a masking policy for customer names.

By default, only SYSADMIN and ACCOUNTADMIN can work with masking policies.

```
USE ROLE SYSADMIN;
CREATE OR REPLACE MASKING POLICY [login]_db.mask_lab.name_mask AS
(val VARCHAR) RETURNS VARCHAR ->
  CASE
    WHEN current_role() IN ('SYSADMIN') THEN regexp_replace(val,'.','*',2)
    WHEN current_role() IN ('TRAINING_ROLE') THEN val
    ELSE '*** REDACTED ***'
    END;
```

18.2.2 Create a masking policy for email addresses.

```
CREATE OR REPLACE MASKING POLICY [login]_db.mask_lab.email_mask AS(val STRING)
   RETURNS STRING ->
  CASE
    WHEN current_role() IN ('TRAINING_ROLE') THEN val
    WHEN current_role() IN ('SYSADMIN') THEN regexp_replace(val,'.+\@','*****@')
    ELSE '*** REDACTED ***'
  END;
```

18.2.3 Create a masking policy for age.

```
CREATE OR REPLACE MASKING POLICY [login]_db.mask_lab.age_mask AS(val INTEGER)
   RETURNS INTEGER ->
  CASE
    WHEN current_role() IN ('TRAINING_ROLE') THEN val
      ELSE null
  END;
```


18.3 Use Data Masking Policies

18.3.1 Set masking policies on the firstname, lastname, email and age columns of the MEMBERS table.

```
ALTER TABLE members MODIFY COLUMN firstname SET MASKING POLICY name_mask;
ALTER TABLE members MODIFY COLUMN lastname SET MASKING POLICY name_mask;
ALTER TABLE members MODIFY COLUMN email SET MASKING POLICY email_mask;
ALTER TABLE members MODIFY COLUMN age SET MASKING POLICY age_mask;
```

18.3.2 As TRAINING_ROLE, view masking policy metadata using the SHOW and DESCRIBE commands.

```
USE ROLE TRAINING_ROLE;

SHOW MASKING POLICIES;
```

18.3.3 DESCRIBE each masking policy.

```
DESC MASKING POLICY name_mask;
DESC MASKING POLICY age_mask;
DESC MASKING POLICY email_mask;
```

18.3.4 View grants on masking policies.

```
SHOW GRANTS ON MASKING POLICY name_mask;
SHOW GRANTS ON MASKING POLICY age_mask;
SHOW GRANTS ON MASKING POLICY email_mask;
```

18.4 Test Masking Policies

18.4.1 Run table queries as the role TRAINING_ROLE.

```
SELECT firstname, lastname, age, email FROM members LIMIT 10;
```

TRAINING_ROLE should be able to see the unmasked data for all columns.

18.4.2 Run table queries as the role SYSADMIN.

```
USE ROLE SYSADMIN;

SELECT firstname, lastname, age, email FROM members LIMIT 10;
```

SYSADMIN should see the first letter of the first and last names, the domain for the email addresses, and NULL for age.

18.4.3 Grant object privileges to the PUBLIC role.

```
GRANT USAGE ON WAREHOUSE [login]_wh TO ROLE PUBLIC;
GRANT USAGE ON DATABASE [login]_db TO ROLE PUBLIC;
GRANT USAGE ON SCHEMA [login]_db.mask_lab TO ROLE PUBLIC;
GRANT SELECT ON ALL TABLES IN SCHEMA [login]_db.mask_lab TO ROLE PUBLIC;
```

18.4.4 Run table queries as the PUBLIC role.

```
USE ROLE PUBLIC;
USE WAREHOUSE [login]_wh;

SELECT firstname, lastname, age, email FROM members LIMIT 10;
```

The PUBLIC role should see NULL for the age column, and ***REDACTED*** or NULL for everything else.

19 Database Replication and Disaster Recovery

This lab will take approximately *40 minutes* to complete.

In this exercise you will learn how to set up two Snowflake accounts for replication, create a primary database, and perform the initial replication of this primary database to a secondary database on another account.

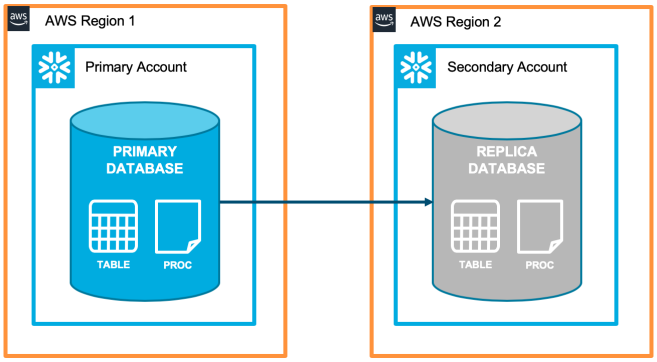

Figure 35: Database Replication Diagram

You will also perform the steps necessary to fail over to the secondary account for disaster recovery.

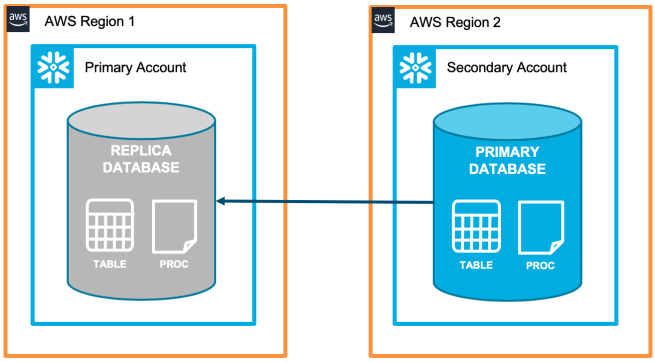

Figure 36: Database Failover Diagram

Before you can configure database replication, two or more accounts must be linked to an organization. The instructor will provide the **Primary** and **Secondary** account URLs for this exercise.

19.1 Set up Browsers for Database Replication

19.1.1 Open two browser windows side-by-side.

You could also open two tabs, but the exercise is easier to complete if you can see both browser windows at the same time.

19.1.2 In browser 1, enter the Primary account URL (provided by your instructor) and log in with your assigned credentials.

19.1.3 Navigate to worksheets and load the script for this lab.

19.1.4 Rename the worksheet to *PRIMARY*.

In the PRIMARY worksheet, you will only execute statements that are surrounded by the PRIMARY comments, as shown below.

```
-- PRIMARY --
-- SQL statement 1;
-- SQL statement 2;
-- PRIMARY --
```

19.1.5 Determine the region and account locator name for your primary account.

```
-- PRIMARY --
SELECT CURRENT_REGION() AS "primary.region",
       CURRENT_ACCOUNT() AS "primary.account_locator";
-- PRIMARY --
```


19.1.6 In browser 2, enter the Secondary account URL and log in with your assigned credentials.

19.1.7 Navigate to worksheets and load the script for this lab.

19.1.8 Rename the worksheet to *SECONDARY*.

In the SECONDARY worksheet, you will only execute statements that are surrounded by the SECONDARY comments, as shown below.

```
-- SECONDARY --
-- SQL command 1;
-- SQL command 2;
-- SECONDARY --
```

19.1.9 Determine the region and account locator names for your secondary account.

```
-- SECONDARY --
SELECT CURRENT_REGION() AS "secondary.region",
    CURRENT_ACCOUNT() AS "secondary.account_locator";
-- SECONDARY --
```

19.1.10 Examine the results.

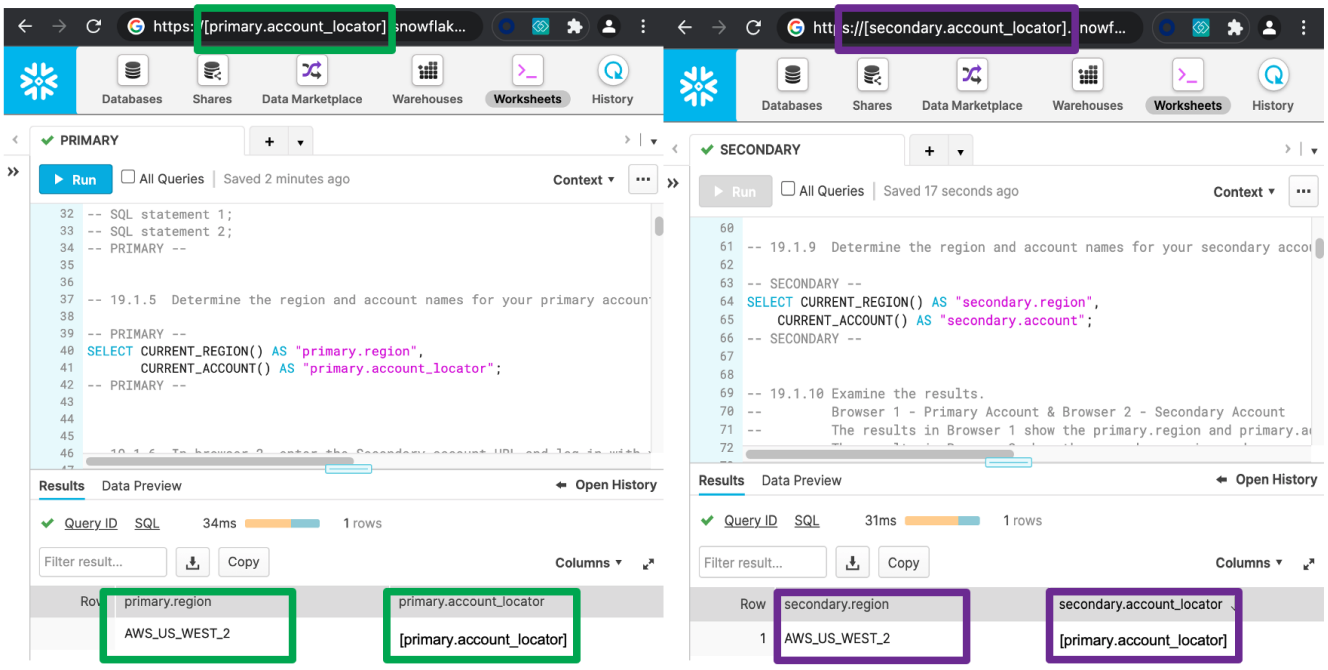

Figure 37: Browser 1 - Primary Account & Browser 2 - Secondary Account

The results in Browser 1 show the **primary.region** and **primary.account_locator**. The results in Browser 2 show the **secondary.region** and **secondary.account_locator**.

For this lab, the primary and secondary accounts are in the same region. However, you can have the secondary account on a different cloud provider, or in a different region, from the primary account.

19.2 Set Account Locator and Region Names

The SQL commands in this lab use an account identifier in the format, `snowflake_region.account_locator`. However, be aware account identifiers in the format `snowflake_region.account_locator` are also supported. For information about account identifiers, see Account Identifiers.

19.2.1 Find and replace account locator and region names.

This lab contains placeholders for the primary and secondary account locators and regions, because those values will be different for every class. Before continuing, you need to replace those placeholders with the correct names.

To do this, use the find/replace feature in the UI. The keyboard shortcut for this feature is:

On MacOS: CMD+OPT+F On Windows: SHIFT+CTRL+F

Using the find/replace feature, find ALL instances of the following placeholders, and replace them with the values returned by the commands you ran on each account

Replace `[PRIMARY-REGION]` with the region for the primary account.

Replace `[PRIMARY-ACCOUNT-LOCATOR]` with the account locator for the primary account.

Replace `[SECONDARY-REGION]` with the region for the secondary account.

Replace `[SECONDARY-ACCOUNT-LOCATOR]` with the account name for the secondary account.

Perform the find/replace this in the script in BOTH the secondary and primary accounts.

19.3 Set Up the Primary Database

19.3.1 On the PRIMARY account, create a database and objects to replicate.

```
-- PRIMARY --
USE ROLE TRAINING_ROLE;

CREATE WAREHOUSE IF NOT EXISTS [login]_REPL_WH
   WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 300;
CREATE DATABASE IF NOT EXISTS [login]_REPL_DB;
CREATE SCHEMA IF NOT EXISTS REPL_SCHEMA;
USE [login]_REPL_DB.REPL_SCHEMA;
```



```
-- create a table with 1000 rows
CREATE OR REPLACE TABLE MARKETING_A
    ( CUST_NUMBER INT, CUST_NAME CHAR(50), CUST_ADDRESS VARCHAR(100),
      CUST_PURCHASE_DATE DATE ) CLUSTER BY (CUST_PURCHASE_DATE)
AS (  SELECT UNIFORM(1,999,RANDOM(10002)),
             UUID_STRING(),
             UUID_STRING(),
             CURRENT_DATE
      FROM TABLE(GENERATOR(ROWCOUNT => 1000))
);

-- create a procedure to insert 100 rows into the table
CREATE OR REPLACE PROCEDURE INSERT_MARKETING_ROWS()
RETURNS VARCHAR NOT NULL
LANGUAGE JAVASCRIPT
EXECUTE AS CALLER
AS
$$
var result = "";
try {
    var sql_command =
        "INSERT INTO MARKETING_A SELECT UNIFORM(1,999,RANDOM(10002)),
           UUID_STRING(), UUID_STRING(), CURRENT_DATE FROM
           TABLE(GENERATOR(ROWCOUNT => 100))"
    stmt = snowflake.createStatement(
        {sqlText: sql_command});
    rs = stmt.execute();
    }
catch (err) {
    result =  "Failed: Code: " + err.code + "\n  State: " + err.state;
    result += "\n  Message: " + err.message;
    result += "\nStack Trace:\n" + err.stackTraceTxt;
    }
return result;
$$
;
-- PRIMARY --
```

19.3.2 View the primary account identifiers.

```
-- PRIMARY --
USE ROLE ACCOUNTADMIN;

SHOW  REPLICATION ACCOUNTS LIKE '[PRIMARY-ACCOUNT-LOCATOR]';
-- PRIMARY --
```

Examine the results. Note the **ACCOUNT_LOCATOR** values and confirm that they match the **Primary** account URL provided by your instructor.

Row	snowflake_region	created_on	account_name	account_locator	comment	organization_name
1	AWS_US_WEST_2	2021-08-05 11:17:17.886 -0700	ACCT588	QDA32321	Class training account f...	SFEDUCATIONALSERVICES2

Figure 38: Show Replication Account Results

Note each value of the **SNOWFLAKE_REGION**, **ACCOUNT_LOCATOR**, **ORGANIZATION_NAME** and **ACCOUNT_NAME** columns to determine the **Primary** account identifiers are in one of the following two formats: `org_name.account_name` or `snowflake_region.account_locator`.

19.3.3 View the secondary account identifiers.

```
-- SECONDARY --
USE ROLE ACCOUNTADMIN;

SHOW  REPLICATION ACCOUNTS LIKE '[SECONDARY-ACCOUNT-LOCATOR]';
-- SECONDARY --
```

Examine the results. Note the **ACCOUNT_LOCATOR** values and confirm that they match the **Secondary** account URL provided by your instructor.

Note each value of the **SNOWFLAKE_REGION**, **ACCOUNT_LOCATOR**, **ORGANIZATION_NAME** and **ACCOUNT_NAME** columns to determine the **Secondary** account identifiers are in one of the following two formats: `org_name.account_name` or `snowflake_region.account_locator`.

19.3.4 Promote **[login]**_REPL_DB to serve as a primary database.

```
-- PRIMARY --
ALTER DATABASE [login]_REPL_DB ENABLE REPLICATION
    TO ACCOUNTS [SECONDARY-REGION].[SECONDARY-ACCOUNT-LOCATOR];
-- PRIMARY --
```

19.3.5 Examine the results of the **SHOW REPLICATION DATABASES** statement.

```
-- PRIMARY --
SHOW REPLICATION DATABASES;
-- PRIMARY --
```

Verify that **IS_PRIMARY** is **TRUE** and that the **REPLICATION_ALLOWED_TO_ACCOUNTS** column contains both the primary and secondary account identifiers in one of the the following two formats: `org_name.account_name` or `snowflake_region.account_locator`.

19.3.6 Enable the ability to fail over from the primary to the secondary account.

```
-- PRIMARY --
ALTER DATABASE [login]_REPL_DB ENABLE FAILOVER
    TO ACCOUNTS [SECONDARY-REGION].[SECONDARY-ACCOUNT-LOCATOR];
```



```
-- PRIMARY --
```

19.3.7 Examine the results of the **SHOW REPLICATION DATABASES** statement.

```
-- PRIMARY --
SHOW REPLICATION DATABASES;
-- PRIMARY --
```

Verify that the **FAILOVER_ALLOWED_TO_ACCOUNTS** column contains both the primary and secondary account identifiers in one of the following two formats: `org_name.account_name` or `snowflake_region.account_locator`.

19.4 Creation and Replication To the Secondary Database

In this exercise, you will perform the steps to create the secondary database, and replicate the database from the primary account to the secondary account.

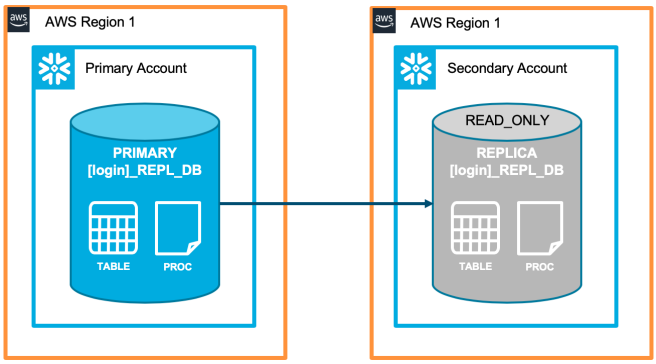

Figure 39: Initial Database Replica

19.4.1 Create a replica database on the secondary account.

This step creates an empty database with the same structure as the database on the primary account. No data will be transferred during this step.

```
-- SECONDARY --
CREATE DATABASE [login]_REPL_DB AS REPLICA
    OF [PRIMARY-REGION].[PRIMARY-ACCOUNT-LOCATOR].[login]_REPL_DB;

SHOW REPLICATION DATABASES;
-- SECONDARY --
```

Examine the results. Check to see that **is_primary** is **FALSE** for the secondary account.

name	comment	is_primary	primary
EFRON_REPL_DB	NULL	false	AWS_US_WEST_2
EFRON_REPL_DB	NULL	true	AWS_US_WEST_2

Figure 40: Show Replication Database Results

19.4.2 Transfer ownership of the replica objects to TRAINING_ROLE.

```
-- SECONDARY --
GRANT OWNERSHIP ON DATABASE [login]_REPL_DB TO ROLE TRAINING_ROLE;
GRANT OWNERSHIP ON SCHEMA [login]_REPL_DB.PUBLIC TO ROLE TRAINING_ROLE;
-- SECONDARY --
```

19.4.3 Start the initial replication.

Replication is a pull operation, so the process is initiated on the secondary account.

```
-- SECONDARY --
USE ROLE TRAINING_ROLE;
ALTER DATABASE [login]_REPL_DB REFRESH;
-- SECONDARY --
```

19.4.4 Query the secondary database to verify the replication has completed.

```
-- SECONDARY --
-- verify data and objects
CREATE WAREHOUSE IF NOT EXISTS [login]_REPL_WH
    WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 300;
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;

SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;

SHOW TABLES;
SHOW PROCEDURES;
-- SECONDARY --
```

The **COUNT(cust_number)** value should be 1000.

19.4.5 Verify that the replica is read-only.

```
-- SECONDARY --
USE WAREHOUSE [login]_REPL_WH;
```



```
CALL INSERT_MARKETING_ROWS();
-- SECONDARY --
```

When the primary database is replicated, a snapshot of its database objects and data is transferred to the secondary database.

19.5 Monitor Replication

In this exercise, you will perform the steps to determine the current status of the initial database replication or a subsequent secondary database refresh.

19.5.1 Set your context.

```
-- SECONDARY --
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_REPL_WH;
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;
-- SECONDARY --
```

19.5.2 Monitor the database refresh progress.

```
-- SECONDARY --
-- show the steps for the latest refresh on the database, in seconds
SELECT
    PHASE_NAME,
    RESULT,
    START_TIME,
    END_TIME,
    DATEDIFF(SECOND, START_TIME, END_TIME) AS DURATION,
    DETAILS
FROM TABLE(INFORMATION_SCHEMA.DATABASE_REFRESH_PROGRESS('[login]_REPL_DB'));

-- show the steps for the latest refresh on the database, in minutes
SELECT value:phaseName::string as Phase,
    value:resultName::string as Result,
    to_timestamp_ltz(value:startTimeUTC::numeric,3) as startTime,
    to_timestamp_ltz(value:endTimeUTC::numeric,3) as endTime,
    datediff(mins, startTime, endTime) as Minutes
FROM TABLE (flatten(input=>parse_json(
    SYSTEM$database_refresh_progress('[login]_REPL_DB'))));
-- SECONDARY --
```

19.5.3 Monitor the database refresh history.

```
-- SECONDARY --
SELECT *
```



```
FROM TABLE(INFORMATION_SCHEMA.DATABASE_REFRESH_HISTORY('[login]_REPL_DB'));
-- SECONDARY --
```

You can also monitor database refresh progress by job id, by providing the value of the JOB_UUID column from the **database_refresh_history** to investigate to a specific refresh in the last 14 days.

```
-- SECONDARY --
SELECT
    PHASE_NAME,
    RESULT,
    START_TIME,
    END_TIME,
    DATEDIFF(SECOND, START_TIME, END_TIME) AS DURATION
FROM TABLE(INFORMATION_SCHEMA.DATABASE_REFRESH_PROGRESS_BY_JOB
    ('<JOB_UUID_VALUE_FROM_REFRESH_HISTORY_QUERY>'));
-- SECONDARY --
```

19.6 Schedule Automatic Refreshes of the Replica

In the previous exercise, you learned how to manually perform the initial refresh and validated the replication between the primary and secondary databases. As a best practice, Snowflake recommends scheduling your secondary database refreshes. In this exercise you will perform the steps for starting a database refresh automatically on a specified schedule.

19.6.1 Create a database on the secondary account where the task will be created.

```
-- SECONDARY --
USE ROLE TRAINING_ROLE;

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

-- create a task to refresh on a regular basis
CREATE OR REPLACE TASK [login]_REPL_DB_REFRESH_TASK
    WAREHOUSE = [login]_REPL_WH
    SCHEDULE = '1 MINUTE'
AS ALTER DATABASE [login]_REPL_DB REFRESH;
-- SECONDARY --
```

19.6.2 Start the task

After creating a task, you must RESUME the task before it will run.

```
-- SECONDARY --
SHOW TASKS;

ALTER TASK [login]_REPL_DB_REFRESH_TASK RESUME;
```



```
SHOW TASKS;
-- SECONDARY --
```

19.6.3 Monitor the task history and the database refresh history.

```
-- SECONDARY --
-- monitor task history
USE WAREHOUSE [login]_REPL_WH;
SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  WHERE DATABASE_NAME LIKE UPPER('[login]_DB');

-- monitor database refresh history
-- LOOK AT ALL REFRESH OPERATIONS FOR THIS DB IN LAST 14 DAYS
SELECT *
   FROM TABLE(INFORMATION_SCHEMA.DATABASE_REFRESH_HISTORY('[login]_REPL_DB'));
-- SECONDARY --
```

19.6.4 Examine the results. This example here shows results after about 5 minutes.

Row	CURRENT_PHASE	START_TIME	END_TIME	JOB_UUID	COPY_BYTES	OBJECT_COUNT
1	COMPLETED	2020-03-24 15:30:...	2020-03-24 15:30:...	01931be6-02f8-9c...	0	10
2	COMPLETED	2020-03-24 15:29:...	2020-03-24 15:29:...	01931be5-0280-bc...	0	10
3	COMPLETED	2020-03-24 15:28:...	2020-03-24 15:28:...	01931be4-02a0-97...	0	10
4	COMPLETED	2020-03-24 15:16:...	2020-03-24 15:18:...	01931bd8-020f-5d...	8192	11
5	COMPLETED	2020-03-24 15:08:...	2020-03-24 15:10:...	01931bd0-0280-46...	71680	11

Figure 41: Monitor Database Refresh History Results

19.7 Verify that a Refresh Picks Up Changes

19.7.1 Insert new rows into the primary database.

```
-- PRIMARY --
USE ROLE TRAINING_ROLE;
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;

CALL INSERT_MARKETING_ROWS();
SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;
CALL INSERT_MARKETING_ROWS();
SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;
-- PRIMARY --
```

COUNT(cust_number) now has the value of 1200.

19.7.2 Check that the secondary database was updated.

The task to refresh the secondary database runs every minute. Wait a minute or so, then use the command below to check the refresh history until you see the new refresh operation start, and then complete.

```
-- SECONDARY --
-- check the database refresh history
SELECT *
  FROM TABLE(INFORMATION_SCHEMA.DATABASE_REFRESH_HISTORY('[login]_REPL_DB'));

-- count the rows in the table
USE WAREHOUSE [login]_REPL_WH;
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;

SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;
-- SECONDARY --
```

The **COUNT(cust_number)** will have a value of 1200 after the refresh completes.

19.7.3 Suspend the Task

```
-- SECONDARY --
USE DATABASE [login]_DB;
USE SCHEMA TASKS;

SHOW TASKS;

ALTER TASK [login]_REPL_DB_REFRESH_TASK SUSPEND;

SHOW TASKS;
-- SECONDARY --
```

19.8 Change Replication Direction

In this exercise, you will perform the steps to promote the secondary database to act as the primary. When promoted, the secondary database becomes writeable. At the same time, the previous primary database becomes a read-only replica database.

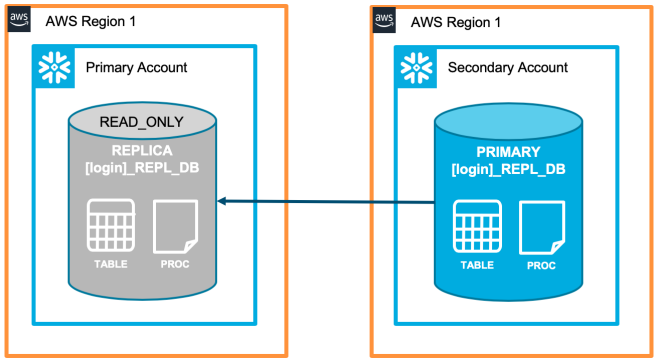

Figure 42: Changing Replication Direction

19.8.1 Promote the secondary database.

```
-- SECONDARY --
-- view replication databases
SHOW REPLICATION DATABASES;

-- fail over to the secondary database
ALTER DATABASE [login]_REPL_DB PRIMARY;
-- SECONDARY --
```

19.8.2 Verify the database on the secondary account is now the primary.

```
-- SECONDARY --
SHOW REPLICATION DATABASES;
-- SECONDARY --
```

Check to see that **is_primary** is **TRUE** for the secondary account.

name	comment	is_primary	primary
EFRON_REPL_DB	NULL	true	AWS_US_WEST_2
EFRON_REPL_DB	NULL	false	AWS_US_WEST_2

Figure 43: Show replication databases results

19.8.3 Verify that the replica database is now writeable.

```
-- SECONDARY --
-- verify database can be written to
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_REPL_WH;
```



```
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;

CALL INSERT_MARKETING_ROWS();
SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;
CALL INSERT_MARKETING_ROWS();
SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;

-- Drop a column from the table
ALTER TABLE MARKETING_A DROP COLUMN CUST_ADDRESS;

DESC TABLE MARKETING_A;

-- recreate the stored procedure without the column that you just dropped
CREATE OR REPLACE PROCEDURE INSERT_MARKETING_ROWS()
RETURNS VARCHAR NOT NULL
LANGUAGE JAVASCRIPT
EXECUTE AS CALLER
AS
$$
var result = "";
try {
    var sql_command =
        "INSERT INTO MARKETING_A SELECT
           UNIFORM(1,999,RANDOM(10002)),UUID_STRING(), CURRENT_DATE FROM
           TABLE(GENERATOR(ROWCOUNT => 100))"
    stmt = snowflake.createStatement(
        {sqlText: sql_command});
    rs = stmt.execute();
    }
catch (err) {
    result =  "Failed: Code: " + err.code + "\n  State: " + err.state;
    result += "\n  Message: " + err.message;
    result += "\nStack Trace:\n" + err.stackTraceTxt;
    }
return result;
$$
;

CALL INSERT_MARKETING_ROWS();

SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;

-- SECONDARY --
```

The **COUNT(cust_number)** value is now 1500.

19.8.4 Refresh the new secondary database.

```
-- PRIMARY --
USE ROLE TRAINING_ROLE;
USE DATABASE [login]_REPL_DB;
USE SCHEMA REPL_SCHEMA;
```



```
SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;

ALTER DATABASE [login]_REPL_DB REFRESH;

-- Wait a minute or so to give the refresh time to complete

DESC TABLE MARKETING_A;

SELECT COUNT(CUST_NUMBER) FROM MARKETING_A;
-- PRIMARY --
```

You should see the record count increase from 1200 before the REFRESH, to 1500 after the REFRESH.

19.9 Clean Up

19.9.1 Run the following statements on the primary account.

```
-- PRIMARY --
USE ROLE TRAINING_ROLE;
DROP DATABASE [login]_REPL_DB;
DROP WAREHOUSE [login]_REPL_WH;
-- PRIMARY --
```

19.9.2 Run the following statements on the secondary account.

```
-- SECONDARY --
USE ROLE TRAINING_ROLE;
DROP DATABASE [login]_REPL_DB;
DROP SCHEMA [login]_db.TASKS;
DROP WAREHOUSE [login]_REPL_WH;
-- SECONDARY --
```

20 Track Table Changes with Streams

This lab will take approximately *25 minutes* to complete.

20.1 Create Basic Table Streams

In this exercise, we will introduce the basic workflow around streams as well as explore the differences between delta and append-only streams.

20.1.1 Navigate to Worksheets and create a new worksheet.

20.1.2 Name the worksheet 'Introduction to Streams'.

20.1.3 Set the Worksheet context as follows:

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

20.1.4 Create a source table.

```
CREATE OR REPLACE TABLE data_staging
(
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_ITEM_SK NUMBER(38,0)
    ,CR_RETURN_QUANTITY NUMBER(38,0)
    ,CR_NET_LOSS NUMBER(7,2)
);
```

20.1.5 Create downstream tables to split the source data for different needs - in this case inventory and cost:

```
CREATE OR REPLACE TABLE data_quantity
(
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_ITEM_SK NUMBER(38,0)
    ,CR_RETURN_QUANTITY NUMBER(38,0)
);

CREATE OR REPLACE TABLE data_cost
(
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_NET_LOSS NUMBER(7,2)
);
```

20.1.6 Create the stream object on the source table:

```
CREATE OR REPLACE STREAM data_check ON TABLE data_staging;
```

20.1.7 Load some sample data into the source table:

```
CREATE WAREHOUSE IF NOT EXISTS [login]_LOAD_WH;
ALTER WAREHOUSE [login]_LOAD_WH SET WAREHOUSE_SIZE = 'MEDIUM';

INSERT INTO data_staging
SELECT CR_ORDER_NUMBER
```



```
      ,CR_ITEM_SK
      ,CR_RETURN_QUANTITY
      ,CR_NET_LOSS
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCDS_SF10TCL"."CATALOG_RETURNS" SAMPLE(5);
```

20.1.8 After the data has loaded, query the table stream object.

```
SELECT
    CR_ORDER_NUMBER
    ,CR_ITEM_SK
    ,CR_RETURN_QUANTITY
    ,CR_NET_LOSS
    ,METADATA$ACTION
    ,METADATA$ISUPDATE
    ,METADATA$ROW_ID
FROM data_check
ORDER BY CR_ORDER_NUMBER LIMIT 10;
```

There are 3 metadata columns included in every row of a stream.

20.2 Query Table Streams

Next we will access the stream and write the data into downstream tables.

20.2.1 Execute statements within a transaction block to consume the stream data.

The stream will reset to a new offset when the transaction is committed.

```
BEGIN;

    INSERT INTO data_quantity (CR_ORDER_NUMBER
                              ,CR_ITEM_SK
                              ,CR_RETURN_QUANTITY)
    SELECT CR_ORDER_NUMBER,CR_ITEM_SK,CR_RETURN_QUANTITY
    FROM data_check t
    WHERE METADATA$ACTION = 'INSERT';

    INSERT INTO data_cost (CR_ORDER_NUMBER,CR_NET_LOSS)
    SELECT t.CR_ORDER_NUMBER,t.CR_NET_LOSS
    FROM data_staging t
    JOIN data_quantity q
        ON t.CR_ORDER_NUMBER = q.CR_ORDER_NUMBER
        AND t.CR_ITEM_SK = q.CR_ITEM_SK;
COMMIT;
```

20.2.2 Take a look at the data in the downstream tables:


```
SELECT * FROM data_quantity;

SELECT * FROM data_cost;
```

20.2.3 Query the table stream to see how it looks after records have been consumed:

```
SELECT * FROM data_check;
```

20.3 Explore Delta and Append Streams

Streams come in two (2) primary varieties, *Delta* and *Append*.

To illustrate the difference you'll create a second source table and stream, populating it with the same records as the original source table.

20.3.1 Create a new source table.

```
CREATE OR REPLACE TABLE data_staging_append
    (
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_ITEM_SK NUMBER(38,0)
    ,CR_RETURN_QUANTITY NUMBER(38,0)
    ,CR_NET_LOSS NUMBER(7,2)
    );
```

20.3.2 Load the same set of data into both source tables:

```
TRUNCATE TABLE data_staging;

INSERT INTO data_staging
    SELECT CR_ORDER_NUMBER,CR_ITEM_SK,CR_RETURN_QUANTITY, CR_NET_LOSS
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCDS_SF10TCL"."CATALOG_RETURNS" SAMPLE(5);

INSERT INTO data_staging_append
    SELECT * FROM data_staging;
```

20.3.3 Create a delta stream on the table data_staging:

```
CREATE OR REPLACE STREAM delta ON TABLE data_staging;
```

A delta stream captures any cumulative changes to the source table.

20.3.4 Create an append stream on the table data_staging_append:

```
CREATE OR REPLACE STREAM append_only ON TABLE data_staging_append APPEND_ONLY=TRUE;
```

An append stream only captures inserts to the source table.

20.3.5 Perform the same UPDATE operation on both tables:

```
UPDATE data_staging
   SET CR_RETURN_QUANTITY = 0
   WHERE CR_RETURN_QUANTITY = 5;

UPDATE data_staging_append
   SET CR_RETURN_QUANTITY = 0
   WHERE CR_RETURN_QUANTITY = 5;
```

20.3.6 Check the status of the table streams:

```
SELECT CR_ORDER_NUMBER
    ,CR_ITEM_SK
    ,CR_RETURN_QUANTITY
    ,CR_NET_LOSS
    ,METADATA$ACTION
    ,METADATA$ISUPDATE
    ,METADATA$ROW_ID
FROM delta
LIMIT 10;

SELECT CR_ORDER_NUMBER
    ,CR_ITEM_SK
    ,CR_RETURN_QUANTITY
    ,CR_NET_LOSS
    ,METADATA$ACTION
    ,METADATA$ISUPDATE
    ,METADATA$ROW_ID
FROM append_only
LIMIT 10;
```

20.3.7 Undo the previous update:

```
UPDATE data_staging
   SET CR_RETURN_QUANTITY = 5
   WHERE CR_RETURN_QUANTITY = 0;

UPDATE data_staging_append
   SET CR_RETURN_QUANTITY = 5
   WHERE CR_RETURN_QUANTITY = 0;
```


20.3.8 Check the status of the streams after the second update.

```
SELECT CR_ORDER_NUMBER
    ,CR_ITEM_SK
    ,CR_RETURN_QUANTITY
    ,CR_NET_LOSS
    ,METADATA$ACTION
    ,METADATA$ISUPDATE
    ,METADATA$ROW_ID
FROM DELTA
LIMIT 10;

SELECT CR_ORDER_NUMBER
    ,CR_ITEM_SK
    ,CR_RETURN_QUANTITY
    ,CR_NET_LOSS
    ,METADATA$ACTION
    ,METADATA$ISUPDATE
    ,METADATA$ROW_ID
FROM append_only
LIMIT 10;
```

20.4 Pair Streams and Tasks

Streams and tasks can be used together to track changes to data over time without the need for manual intervention.

20.4.1 Navigate to Worksheets and create a new worksheet.

20.4.2 Name the worksheet 'Streams & Tasks Together'.

20.4.3 Set the Worksheet context as follows:

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

20.4.4 Create a basic data staging (source) table with a table stream and downstream tables to use for the exercise:

```
CREATE OR REPLACE TABLE data_staging
(
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_ITEM_SK NUMBER(38,0)
    ,CR_RETURN_QUANTITY NUMBER(38,0)
    ,CR_NET_LOSS NUMBER(7,2)
);
```



```
CREATE OR REPLACE STREAM data_check ON TABLE data_staging;
```

20.4.5 Create downstream tables to use for the exercise:

```
CREATE OR REPLACE TABLE data_quantity
(
     CR_ORDER_NUMBER NUMBER(38,0)
    ,CR_ITEM_SK NUMBER(38,0)
    ,CR_RETURN_QUANTITY NUMBER(38,0)
);

CREATE OR REPLACE TABLE data_cost
(
    CR_ORDER_NUMBER NUMBER(38,0)
   ,CR_NET_LOSS NUMBER(7,2)
);
```

20.4.6 Create a stored procedure that performs your transformations.

The stored procedure moves data downstream from our staging table.

```
CREATE OR REPLACE PROCEDURE usp_load_prod()
RETURNS STRING NOT NULL
LANGUAGE javascript
AS
$$
  var my_sql_command = ""

  var my_sql_command = "INSERT INTO data_quantity \
      (CR_ORDER_NUMBER,CR_ITEM_SK,CR_RETURN_QUANTITY) \
      SELECT CR_ORDER_NUMBER,CR_ITEM_SK,CR_RETURN_QUANTITY \
      FROM data_check t \
      WHERE METADATA$ACTION = 'INSERT'";

  var statement1 = snowflake.createStatement( {sqlText: my_sql_command} );
  var result_set1 = statement1.execute();

  var my_sql_command = "INSERT INTO data_cost (CR_ORDER_NUMBER,CR_NET_LOSS) \
                        SELECT t.CR_ORDER_NUMBER,t.CR_NET_LOSS \
                        FROM data_staging t \
                          JOIN data_quantity q \
                        ON t.CR_ORDER_NUMBER = q.CR_ORDER_NUMBER \
                        AND t.CR_ITEM_SK = q.CR_ITEM_SK;";

  var statement2 = snowflake.createStatement( {sqlText: my_sql_command} );
  var result_set2 = statement2.execute();

  return my_sql_command; // Statement returned for info/debug purposes
$$;
```


20.4.7 Create a task to run the stored procedure on a regular basis.

The task will run on a schedule. If it finds data in the table stream, it will execute the stored procedure:

```
CREATE OR REPLACE TASK capture_data
  WAREHOUSE = ANIMAL_TASK_WH
  SCHEDULE = 'USING cron * * * * * UTC'
WHEN
  SYSTEM$STREAM_HAS_DATA('data_check')
AS
  CALL usp_load_prod();

ALTER TASK capture_data RESUME;

SHOW TASKS;
```

20.4.8 To test the task, insert some data into the staging table:

```
INSERT INTO data_staging
  SELECT CR_ORDER_NUMBER,CR_ITEM_SK,
  CR_RETURN_QUANTITY, CR_NET_LOSS
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCDS_SF10TCL"."CATALOG_RETURNS" SAMPLE(5);

SELECT * FROM data_check LIMIT 20;
```

20.4.9 After a couple of minutes, check the downstream tables to verify the task executed:

```
SELECT TOP 10 * FROM data_quantity;

SELECT TOP 10 * FROM data_cost;
```

20.4.10 Finally, suspend the task to avoid any unwanted credit spending:

```
ALTER TASK capture_data SUSPEND;
```

