3 Work With Database Objects

3.1 Lab Introduction

3.1.1 Purpose

The purpose of this lab is to familiarize you with some of the objects you'll encounter in Snowflake. Specifically, you'll learn about permament tables, transient tables, and temporary tables. You'll also learn how to create and use auto-increment fields. Finally, you'll learn how to work with views.

3.1.2 What you'll learn

- How to create and work with temporary tables.
- How to create an auto-increment column
- How to create views

3.1.3 How to complete this lab

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

You can also use the SQL code file for this lab that was provided at the start of the class. You would simply need to open it in TextEdit (Mac) or Notepad (Windows), and then copy and paste the SQL code directly into a worksheet.

3.2 Work With Permanent, Temporary, and Transient Tables

3.2.1 Create a worksheet named *Work with Objects*, and set the context.

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE LEARNER_WH;
USE DATABASE LEARNER_DB;
USE SCHEMA PUBLIC;
```

3.2.2 Create a table named **permanent**, with three columns (ID, first name, and last name). Make ID an INTEGER, first name a VARCHAR(20), and last name a VARCHAR(30):

```
CREATE TABLE permanent (id INT, first VARCHAR(20), last VARCHAR(30));
```


3.2.3 Create a temporary table named **temp**, with the same three columns:

```
CREATE TEMPORARY TABLE temp
(id INT, first VARCHAR(20), last VARCHAR(30));
```

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

```
CREATE TRANSIENT TABLE transient
(id INT, first VARCHAR(20), last VARCHAR(30));
```

3.2.5 Show the tables with SHOW TABLES. Notice that the “kind” column tells you the type of each table:

```
SHOW TABLES;
```

3.2.6 Open another worksheet. Copy these commands into the new worksheet and run them to show the tables:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE LEARNER_WH;
USE DATABASE LEARNER_DB;
USE SCHEMA PUBLIC;

SHOW TABLES;
```

3.2.7 In the new worksheet SHOW TABLES does not list the temporary table since it is tied to an entirely different session, but the transient table is listed. Now close the new worksheet and drop the transient table in the original worksheet (this one):

```
DROP TABLE transient;
```

3.2.8 Add three rows of data to your **permanent** table:

```
INSERT INTO permanent
VALUES (1, 'Jarit', 'Johnson'),
(2, 'Shayla', 'Nguyen'),
(3, 'Peewee', 'Herman');
```

3.2.9 Query the table to verify the columns and data.

```
SELECT * FROM permanent;
```


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

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

3.2.11 Query the table to see what values are in the **comments** column:

```
SELECT * FROM permanent;
```

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

```
DESCRIBE TABLE permanent;
```

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

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

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

3.2.14 DROP the comments column, then re-add it as VARCHAR(1000):

```
ALTER TABLE permanent DROP COLUMN comments;
ALTER TABLE permanent ADD COLUMN comments VARCHAR(1000);
```

3.2.15 DESCRIBE the table and verify the column definition is correct:

```
DESCRIBE TABLE permanent;
```

3.2.16 Create a table from SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS, using $ notation to select columns 1, 3, 4, 5, and 6.

3.2.17 Use AS to rename the columns.

3.2.18 Name it **my_orders**. It will take a few minutes to create:


```
CREATE TABLE my_orders AS
SELECT $1 AS key, $3 AS status, $4 AS price, $5 AS date, $6 AS priority
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS;
```

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

```
SELECT * FROM my_orders
LIMIT 10;
```

3.2.20 Create a table using LIKE:

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

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

```
SELECT * FROM like_orders;
DROP TABLE like_orders;
```

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

3.3 Create a table with an auto-increment column and a default column

3.3.1 First, create the table below. Note that the part number column auto-increments and that the restocking fee column has a default value of 5.50.

```
CREATE OR REPLACE TABLE autoincDefault_demo (part_num integer AUTOINCREMENT,
   part_name varchar(30), restocking_fee number(8,2) DEFAULT 5.50 );
```

3.3.2 Now let's insert some data and check out the result:

```
INSERT INTO autoincDefault_demo (part_name) values ('Wheel'), ('Tires');

SELECT * FROM autoincDefault_demo;
```

3.3.3 Note that there are two rows and the part numbers are 1 and 2 respectively. Because we didn't insert a value for the restocking fee, the value for restocking fee for both rows is 5.50.

3.3.4 Now let's insert two more rows and provide a specific value for restocking fee:


```
INSERT INTO autoincDefault_demo (part_name, restocking_fee) values ('Engine
   Block', 25), ('Transmission', 75.50);

SELECT * FROM autoincDefault_demo;
```

3.3.5 As you can see, the part number values for the new rows are 3 and 4 respectively. Snowflake gave the rows those values because that column is set to auto-increment. The restocking fee is overwritten since those values were provided.

3.4 Work with Views

3.4.1 Set the context and create a view from the **my_orders** table you created in the previous task:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE LEARNER_WH;
USE DATABASE LEARNER_DB;
USE SCHEMA PUBLIC;

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

3.4.2 Create a secure view with the same information:

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

3.4.3 Create a materialized view with the same information:

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

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

3.4.4 SHOW your views. Examine the columns to determine how you can tell what type of view it is:

```
SHOW VIEWS;
```

3.4.5 See if you can create a view that is both secure and materialized:


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

INSERT INTO members
VALUES
(103, 'Barbra', 'Streisand', '10/05/2019', 'silver'),
(95, 'Ray', 'Bradbury', '06/06/2006', 'bronze'),
(111, 'Daenerys', 'Targaryen', '2/4/2019', 'gold'),
(87, 'Homer', 'Simpson', '3/1/1998', 'gold');

CREATE SECURE MATERIALIZED VIEW sm_view
  AS SELECT first_name, last_name FROM members;
```

3.4.6 Select all data from your **orders_view** view. Note how long it takes:

```
SELECT * FROM orders_view;
```

3.4.7 Suspend your warehouse to clear any remaining cache:

```
ALTER WAREHOUSE LEARNER_WH SUSPEND;
```

3.4.8 Select all data from your **s_orders_view** view, and note how long it takes:

```
SELECT * FROM s_orders_view;
```

3.4.9 Suspend your warehouse then select all data from your **m_orders_view** view. Note how long it takes:

```
ALTER WAREHOUSE LEARNER_WH SUSPEND;
SELECT * FROM m_orders_view;
```

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

3.4.10 Drop your views and suspend the warehouse:

```
DROP VIEW orders_view;
DROP VIEW s_orders_view;
DROP MATERIALIZED VIEW m_orders_view;
DROP MATERIALIZED VIEW sm_view;
ALTER WAREHOUSE LEARNER_WH SUSPEND;
```


3.5 Key Takeaways

- SHOW TABLES only lists temporary tables that exist in the current session.
- SHOW TABLES can list transient tables across sessions.
- You can use ALTER TABLE to add or drop columns from an existing table.
- DESCRIBE TABLE can be used to show the table definition.
- You can create tables based on the definition of another table but without copying the data.
- You can put AUTOINCREMENT after a integer field's data type in a table's CREATE statement to create an auto-increment column.
- SHOW VIEWS will show you a list of views and their types.

