7 Introduction to Tasks

7.1 Lab Introduction

7.1.1 Purpose

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

7.1.2 What you'll learn

- How to create a task
- How to create a tree of tasks
- How to start and stop a tree of tasks
- How to alter a task with a stored procedure
- How to call a stored procedure from a task

7.1.3 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.

7.2 Scenario

In this exercise you will create a tree of four tasks. The scenario is that for all orders in a specific month, you need to capture and store the average number of days from the time an order is created until it is shipped. The idea is that the table would be updated once a month for reporting purposes.

The first task you create will generate a subset of data from the orders and lineitem tables and store that in a table. In the real world this data would be raw data generated inside Snowflake or loaded into Snowflake. For the purpose of this exercise we will generate this data from the order and lineitem tables.

The second task will be a sub-task that averages that data and stores it in a table. In the real world this would be the final table used for reporting purposes.

The third task will be a sub-task that generates rows containing year and month values that the first and second tasks will use to generate their data. Although this lab is intended to simulate a monthly reporting cycle, for obvious reasons you will fire the first (main) task once a minute rather than once a month in order to aggregate the monthly data.

A fourth task will stop the main task after it runs 10 times. This is to ensure that the tasks aren't left running inadvertently after the lab is completed. At the end of the lab, you will also be asked to delete all tables and tasks for the same purpose.

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

Let's get started!

7.3 Setting the context

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

7.3.1 Run the commands below to set the context.

```
USE ROLE TRAINING_ROLE;
CREATE DATABASE IF NOT EXISTS LEARNER_DB;
CREATE WAREHOUSE IF NOT EXISTS LEARNER_WH;
USE WAREHOUSE LEARNER_WH;
ALTER WAREHOUSE LEARNER_WH SET WAREHOUSE_SIZE = xsmall;
USE DATABASE LEARNER_DB;
CREATE SCHEMA IF NOT EXISTS LEARNER_TASKS_SCHEMA;
USE SCHEMA LEARNER_TASKS_SCHEMA;
```

7.4 Creating the tables

Now let's create the tables we need.

7.4.1 Run the following command to create the final reporting table that will hold the average days to ship:

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

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

Now let's create the table that will hold the transformed data from the order and lineitem tables that will be used to populate the previously created table.

7.4.2 Run the following command to create the table that will hold the transformed data:

```
CREATE OR REPLACE TABLE SHIPPING_BY_DATE (orderdate DATE, shipdate DATE,
   daystoship DECIMAL(18,2));
```

As you can see, we will pull over only the order date, the shipping date, and the difference between them.

Now let's create one more table. We'll need it to store rows, each of which will contain unique year and month values. The tasks will fetch the latest month and year from the table in order to know which set of data to transform and load into the previously created table. We'll also insert the first row into the table. One of our tasks will automate the insertion of subsequent rows.

7.4.3 Run the commands below to create the year_month table and insert the first required row of data and to verify the insert took place:

```
CREATE OR REPLACE TABLE year_month(id integer autoincrement(1,1), yr INTEGER, mon
   INTEGER);

INSERT INTO year_month(yr, mon) VALUES (1992, 1);

SELECT * FROM year_month;
```

7.5 Creating the tasks

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

7.5.1 Run the following command to create the task that will load the raw data table with data from the orders and lineitem tables:

```
CREATE OR REPLACE TASK insert_shipping_by_date_rows
    WAREHOUSE = 'LEARNER_WH'
    SCHEDULE = 'USING CRON 0-59 0-23 * * * America/Chicago'
AS
    INSERT INTO LEARNER_DB.LEARNER_TASKS_SCHEMA.SHIPPING_BY_DATE (ORDERDATE,
       SHIPDATE, DAYSTOSHIP)
    SELECT
        O_ORDERDATE
        , L_SHIPDATE
        , L_SHIPDATE - O_ORDERDATE AS FULFILL_DATE
    FROM
        TRAINING_DB.TPCH_SF1.ORDERS O LEFT JOIN TRAINING_DB.TPCH_SF1.LINEITEM L ON
           O.O_ORDERKEY = L.L_ORDERKEY
    WHERE
        YEAR(O_ORDERDATE) = (SELECT yr FROM
           LEARNER_db.LEARNER_tasks_schema.year_month WHERE ID = (select MAX(ID)
           from year_month))
        AND
        MONTH(O_ORDERDATE)= (SELECT mon FROM
           LEARNER_db.LEARNER_tasks_schema.year_month WHERE ID = (select MAX(ID)
           from year_month));
```

Notice that the schedule is set to run every minute of every hour. For the purposes of this lab, each minute will bring over one month's worth of data from the order and lineitem tables.

Note that the AS clause of the task contains an INSERT statement that calls a SELECT statement to populate the table. The WHERE clause of the SELECT statement that the INSERT statement uses to populate the table also selects data where the year and month fields' values match the month and year values most recently inserted into the year_month table. Thus, each minute it will pull one month's worth of data.

7.5.2 Run the following commands to create the task that will populate the table for the average monthly shipping:

```
CREATE OR REPLACE TASK average_monthly_shipping
    WAREHOUSE = 'LEARNER_WH'
    AFTER insert_shipping_by_date_rows
AS
    INSERT INTO LEARNER_DB.LEARNER_TASKS_SCHEMA.AVG_SHIPPING_IN_DAYS(yr, mon,
       avg_shipping_days)
    SELECT
        YEAR(orderdate) AS yr
      , MONTH(orderdate) AS mon
      , AVG(daystoship) AS avg_shipping_days
    FROM
        LEARNER_DB.LEARNER_TASKS_SCHEMA.SHIPPING_BY_DATE
    WHERE
        yr = (SELECT yr FROM LEARNER_db.LEARNER_tasks_schema.year_month WHERE ID =
           (select MAX(ID) from year_month))
        AND
        mon = (SELECT mon FROM LEARNER_db.LEARNER_tasks_schema.year_month WHERE ID
           = (select MAX(ID) from year_month))
    GROUP BY
        YEAR(orderdate)
      , MONTH(orderdate);
```

Note that this task does not have a schedule but rather an AFTER clause so it executes after the main task completes. The WHERE clause of the SELECT statement that the INSERT statement uses to populate the table also selects data where the yr and mon fields' values matches the month and year values most recently inserted into the year_month table.

7.5.3 Run the following command to create the task that will increment the year and month in the year_month table:

```
CREATE OR REPLACE TASK increment_year_month
    WAREHOUSE = 'LEARNER_WH'
    AFTER average_monthly_shipping
AS
    INSERT INTO year_month (yr, mon)
    SELECT DISTINCT
        IFF(mon=12, yr+1, yr) AS YR
      , IFF(mon=12, 1, mon+1) AS MON
    FROM year_month
    WHERE ID = (select MAX(ID) from year_month);
```

This task is fairly simple. As you can see from the AFTER clause and the SELECT statement, after the previous task averages the current month's monthly shipping time frame in days, a new row is added for the next month.

7.5.4 Run the following commands to create the task that will stop the tree from executing:

```
CREATE OR REPLACE PROCEDURE stop_tasks()
 RETURNS string
 LANGUAGE javascript
 STRICT
AS
$$
    var r='continue';
    var mon = snowflake.createStatement({sqlText: 'SELECT mon FROM
       LEARNER_db.LEARNER_tasks_schema.year_month WHERE ID = (select MAX(ID) from
       year_month);'});

    var result = mon.execute();
    result.next();
    var x = result.getColumnValue(1);

    if (x>=11) {

    var killstmt1 = snowflake.createStatement({sqlText: 'ALTER TASK
       insert_shipping_by_date_rows SUSPEND;'});
    var killstmt2 = snowflake.createStatement({sqlText: 'ALTER TASK
       average_monthly_shipping SUSPEND;'});
    var killstmt3 = snowflake.createStatement({sqlText: 'ALTER TASK
       increment_year_month SUSPEND;'});

    killstmt1.execute();
    killstmt2.execute();
    killstmt3.execute();

    r = 'stopped'
    }
 return r;

$$;

CREATE OR REPLACE TASK stop_tasks
    WAREHOUSE = 'LEARNER_WH'
    AFTER increment_year_month
AS
    CALL stop_tasks();
```

The idea behind this task is that it will call a stored procedure after all the other tasks have fired and all the required inserts have taken place. The stored procedure will suspend the previous three tasks if there is a month in any of the year_month table rows that has an 11. This will keep the tasks from running indefinitely.

7.6 Starting the tree

Now we're going to start the tree and monitor the activity.

7.6.1 Run the following commands to start the tree:

```
SELECT system$task_dependents_enable('insert_shipping_by_date_rows');
```

This system function recursively starts all the tasks in the tree. If you were starting only one task you would use an ALTER TASK statement to start it.

7.6.2 Use the following queries to monitor what is going on in the tables over the next 10-15 minutes and to view the state of the tasks:

```
SELECT * FROM year_month;
SELECT * FROM SHIPPING_BY_DATE;
SELECT * FROM avg_shipping_in_days;

DESCRIBE TASK insert_shipping_by_date_rows;
DESCRIBE TASK average_monthly_shipping;
DESCRIBE TASK increment_year_month;
```

You should notice that year and month rows are being added to the year_month table, that the SHIPPING_BY_DATE table is full of raw data, and that the avg_shipping_in_days table is getting a new month of aggregated data each minute.

7.7 Garbage Collection

7.7.1 If you don't wish to wait until the last task has stopped the process and are ready to complete the lab, you can run the ALTER TASK command below to immediately stop the task tree:

```
ALTER TASK insert_shipping_by_date_rows SUSPEND;
```

7.7.2 If you waited until the last task in the tree stopped the process, or if you are done with the lab and ran the previous ALTER TASK command, run the commands below to clear out the objects used in this lab:

```
DROP TABLE shipping_by_date;
DROP TABLE avg_shipping_in_days;
DROP TABLE year_month;
DROP TASK insert_shipping_by_date_rows;
DROP TASK average_monthly_shipping;
DROP TASK increment_year_month;
DROP TASK stop_tasks;
DROP SCHEMA LEARNER_TASKS_SCHEMA;
ALTER WAREHOUSE LEARNER_WH SUSPEND;
```

7.8 Key Takeaways

- The first task in a tree of tasks can be started with a CRON statement.

- Subsequent tasks in a tree of task can be started by calling them from previous tasks with an AFTER clause.
- You can start a tree of tasks with this statement: SELECT system$task_dependents_enable('<>');.
- It is imperative to stop a task you no longer need in order to avoid wasting credits.
- A stored procedure can be used to stop a tree of tasks.
- You can use the statement CALL stored_procedure_name() in the AS clause of a task to invoke a stored procedure.