Experiment: Data Engineering Pipeline

Overview

This experiment demonstrates Snowflake features specifically aligned to “Data Engineering” workload to build modern data pipelines. Data pipelines automate many of the manual steps involved in transforming and optimizing continuous data loads. Frequently, the “raw” data is firstloaded temporarily into a staging table used for interim storage and then transformed using a series of SQL statements before it is inserted into the destination reporting tables. The most efficient workflow for this process involves transforming only data that is new or modified.

Experiment 24: Tasks

At this point, 2 of the 3 pieces of the data engineering pipeline are in place. The last step is to create the TASKS that will operate on the STREAMS you just created. Tasks allow us to schedule and orchestrate ETL logic which consumes the records collected by the streams and runs DML transactions (insert, merge) on those records. The completion of the DML transaction resets the stream so we can capture changes going forward from here.

Tasks need a warehouse to execute their DML commands so create a warehouse specifically to handle this. In a production environment you could choose to create a dedicated warehouse like here or use an existing ETL warehouse. In the examples below we will use the existing DATAPIPELINES_WH to execute TASK activities.

24.1 Create and Activate Tasks

24.1.1 Create the push_trips task to read JSON data in the streams_trips stream and transform and write it to the trips table you just created. You indicate the warehouse where this task will run and how frequently it will run. You can also add the optimization to only run this task if there is new data in the stream to process. In the event no new data has arrived in the last minute, the task will not kick off and you can potentially save the startup cost for the warehouse.

```
create or replace task
push_trips warehouse =
DATAPIPELINES_WH schedule
= '1 minute'
when system$stream_has_data('stream_trips')
as
insert into bike_trips
  select
  v:tripduration::integer,
  v:starttime::timestamp_nt
  z,
  v:stoptime::timestamp_ntz
  ,
  v:start_station_id::integ
  er,
  v:end_station_id::integer
  , v:bikeid::integer,
  v:usertype::string
  from stream_trips;
```

24.1.2 Create a TASK to perform a merge into the stations table.

```
create or replace task
push_stations warehouse =
DATAPIPELINES_WH schedule =
'1 minute'
when
system$stream_has_data('stream_station
s') as
merge into
  bike_stations s
  using (
    select v:start_station_id::integer
      station_id, v:start_station_name::string
      station_name,
      v:start_station_latitude::float
      station_latitude,
      v:start_station_longitude::float
      station_longitude,
      'Station at ' || v:start_station_name::string
    station_comment from stream_stations
    union
    select v:end_station_id::integer
      station_id, v:end_station_name::string
      station_name,
      v:end_station_latitude::float
      station_latitude,
      v:end_station_longitude::float
      station_longitude,
      'Station at ' || v:end_station_name::string
    station_comment from stream_stations) ns
  on s.station_id =
  ns.station_idwhen not
  matched then
    insert (station_id, station_name, station_latitude,
station_longitude,station_comment)
    values (ns.station_id, ns.station_name,
ns.station_latitude,ns.station_longitude,
ns.station_comment);
```

24.1.3 You can chain TASKS together to form sophisticated processing operations. Define a TASK to call the purge_files procedure AFTER the push_trips TASK runs.

```
create or replace task
purge_files  warehouse =
DATAPIPELINES_WH after
push_trips
as
  call purge_files('trips_raw', '@streaming_data/');
```

24.1.4 Tasks need to be activated after they are created or anytime they are modified in some way. Activate the tasks that were just created.

```
alter task purge_files resume;

alter task push_trips resume;
alter task push_stations resume;

show tasks;
```

24.1.5 Verify data loaded from streams to the tables.

```
select count(*) from
citibike_pipelines.public.bike_trips;

select count(*) from
citibike_pipelines.public.bike_stations;
```