Hive ACID Labs

These are labs to accompany Cloudera SKO 2020 Cross-training in Hive ACID.

Preparation

1. Cluster Access - These labs are designed for Hive 3.x, so you will need access to an HDP 3.x cluster or a CDP cluster with Hive installed. For the SKO sessions, we will provide a shared cluster. If you are doing this separately, you can spin up a cluster through Squadron, CDP demos, or whatever method you prefer. You can even use one of the product “Sandbox” downloads.

2. Github Repo - For your convenience DDL’s (and finished scripts if you are lazy!) are available in a github repository. Note: The sql in these labs can be found in sql/labs.sql, and it is usually easier to copy and paste from a text file to avoid weird quote characters, etc. that word processors like to add.
   git clone https://github.com/screamingweasel/acid

3. Preparing your environment - After logging into beeline or hive on your cluster
   a. Run script **ddl/people.ddl** from the repo and pass a unique db name. This will create a unique database as well as the people and people_type2 tables.
      hive --hivevar db=<mydb> -f ddl/people.ddl

4. Load Data - For the SKO session we will have already loaded a source table default.people_raw. If you are doing this on your own you can use the script load-people-raw.sh to load this sample data into your cluster.
   ./load-people-raw.sh

The sample data is a set of 1,000 people records, keyed by an integer id. We will be using subsets of these rows to perform the dml operations.

5. Describe your people table and take note of the file location
```
use <mydatabase>;
show create table people;

-- Example Location
‘hdfs://c316-node2.squadron.support.hortonworks.com:8020/warehouse/table
space/managed/hive/jbarnett.db/people’
```

For many of the labs we will be looking at the Hive table’s files to see the effects of DML and compactions. You can do this easily from within the beeline shell using the !sh command:

```
!sh hdfs dfs -ls -R
hdfs://c316-node2.squadron.support.hortonworks.com:8020/warehouse/tables
pace/managed/hive/jbarnett.db/people

# You can also dump and look at the data with the hive orc dumpfile utility
hive --orcfiledump -d \
hdfs://c316-node2.squadron.support.hortonworks.com:8020/warehouse/tablespace/ma
naged/hive/foo.db/people/delete_delta_0000002_0000002_0000
```

Lab 1 - Simple Insert

Insert 10 records from the default.people_raw into your people table where id between 1 and 10

Lab 2 - Update

Update rows with id 1 through 5 by concatenating a suffix to the first and last names.

Lab 3 - Additional Inserts

A. Insert new rows from the people_raw table where id between 11 and 20
B. Insert several rows using hard-coded insert values() statements

Note that the bulk inserts were performed in a single transaction; whereas the individual inserts each had their own transaction.

Lab 4 - Deletes

Delete people table rows where id in (1,3,5)

Lab 5 - Compaction

A. List the files in the people table
B. Perform a minor Compaction and wait until compaction is complete
```
Alter table people compact ‘minor’;
-- Repeat until completed
Show compactions;
```
C. Take note of the file changes
D. Perform the same steps with a major compaction

Lab 6 - Merge

Write a sql merge statement with the WHEN MATCHED THEN UPDATE and WHEN NOT MATCHED THEN INSERT statements. For updates, select from default.people_raw where id between 11 and 20 and concatenate some unique suffix to the first and last names. Then use a union to select from default.people_raw where id between 21 and 30. Join key is id.

After running the merge, take a look at the files, and if you have time run a compaction.

Lab 7 - SCD Type 2 Merge (Optional)

Many applications keep a history of changes made to rows and if a row is deleted they will flag it as deleted instead of physically deleting the rows. This is a common pattern in data warehousing and is known as a Slowly Changing Dimension Type 2 (SCD 2) pattern. Hive merge supports this very well.

Earlier you should have created a people_type2 table. The unique key of this is id + begin_dt. The end_dt is set to '9999-12-31' to indicate the current row, and if this row has a lower date then the row is considered deleted. There are other ways to represent this (current_flag, deleted_flag) but this is the simplest.

There are a couple of ways to go about this, but all of these involve a union clause in the USING clause. You need to generate an INSERT row for each insert and update (a new version) and an UPDATE row to set the end_dt to current_date - 1 for each updated or deleted row.

One of the canonical tutorials on this can be found at:
https://github.com/cartershanklin/hive-scd-examples