Lab 2: Hive Commands

These labs assume you are using the class repository from the Nuvelab VM. You can also clone the repo to your HDP VM but that variation is not covered in the lab instructions.

The commands used in this lab are available in the text file “commands.txt” in the lab directory.

Data Setup

1. Log into the Ambari file explorer
2. Create a new lab2 directory in /user/maria_dev that is fully writeable, just like you did in lab1.
3. Make sure still have the /user/maria_dev/data you created in the last lab. If not, recreate it using the instructions in lab1.
4. Make sure the git repo is up to date but executing the command “git pull” in the Nuvelab VM.
5. Upload the sales.csv file into the lab2 directory and make it fully writeable like you did for the CSV file in lab1.

Creating the First Table

The schema for the table is:

```
orderdate   string
region      string
rep         string
item        string
units integer
unitcost    double
total double
```

1. Create a database called “salesdept” and switch to it. If you can’t remember how to do this, review lab 1. Confirm there are no tables in the database

2. Create an internal table sales1

```
create table sales1(
      orderdate   string,
      region      string,
      rep         string,
      item        string,
      units       int,
      unitcost    double,
      total       double)
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;
```

3. If you are getting unusual parse errors, check to make sure that the single quote characters are the ascii single quote (') and not the open and close quote (‘’) characters which word processors use.
4. Set the table properties to skip the first row on input.

```
alter table sales1 SET TBLPROPERTIES ("skip.header.line.count"="1");
```

5. Also set the hive cli property to show the column headers

```
set hive.cli.print.header=true;
```

6. Load the data into the table and confirm the table is loaded;

```
load data inpath'/user/maria_dev/lab2/sales.csv' into table sales1;

select * from sales1 limit 20
```

Table Insert Selected Columns

1. Create new table

```
create table inventory(
      product     string,
      amount      int,
      price       double)
stored as textfile;
```

2. Use the show tables command to ensure that the table is there and use the select command to confirm the table is empty.
3. Insert data from the sales1 table into the inventory table. Only three columns are going to be selected from the sales table for inclusion into the inventory table. Note that we

don't have to alter the inventory table since skipping the first header row was already done when we loaded the data into the sales table:

```
insert into table inventory select item, units, unitcost from sales1;
```

4. Confirm the insert by doing

```
select * from inventory;
```

5. Use the formatted describe command to see how many rows there are:

```
describe formatted inventory;
```

6. Run the inset command from step 3 again and then run the formatted describe command from step 5.

```
insert into table inventory select item, units, unitcost from sales1
describe formatted inventory;
```

7. Notice that there are now twice as many rows in the inventory table than in the sales1 table. The second insert command appended the values to the end of the table
8. Now use the insert overwrite command to delete the existing data from the inventory table and overwrite it with new data.

```
insert overwrite table inventory select item, units, unitcost from sales1;
```

9. Once you have done this, use describe formatted command from step 5 to confirm there are only 43 rows.

Table Insert Selected Rows

1. We can selectively insert rows from the sales1 database into the inventory database by using a where clause

```
insert overwrite table inventory select item, units, unitcost
      from sales1 where item = 'Pen';
```

2. Confirm using the select command that the inventory table contains only entries for pens.

3. Create two new tables called pencils and binders using the same definition as for inventory. Notice that the schema is different for the two tables.

```
create table pencils(
      product     string,
      amount      int,
      price       double)
stored as textfile;


create table pencils(
      product     string,
      amount      int,
      price       double)
stored as textfile;
```

4. Use show the tables and the select command to verify the tables exist and are empty.
5. Now we can distribute the contents of the sales1 table to the pencils and binders tables. Notice that the syntax is different than the previous insert commands we used

```
from sales1
 insert into pencils select item, units, unitcost where item = 'Pencil'
 insert into binders select item, units, unitcost where item = 'Binder';
```

6. Verify that the tables are both correctly populated by pencil and binder entries.
7. We can also use logical operators in our where clause. Modify the insert statement from step 1 to read:

```
insert overwrite table inventory select item, units,
      unitcost from sales1 where item = 'Pen' or item = 'Pencil';
```

Altering Table Schema

1. Try to overwrite the inventory table selecting a different number of columns from sales1 than there are columns in inventory

```
insert overwrite table inventory select item, units, unitcost,total
      from sales1;

insert overwrite table inventory select item, units from sales1;
```

2. Notice that the command fails. However, if you try it with mismatched column types, the command works, but NULLS are inserted when there is a type conversion issue. Experiment with the following commands to see what the resulting inventory table

looks likes after executing the three commands below. What is the general rule you can deduce from these examples for how mismatched data types are handled?

```
insert overwrite table inventory select unitcost, unitcost, unitcost
      from sales1;

insert overwrite table inventory select units, units, units from sales1;

insert overwrite table inventory select item, item, item from sales1;
```

3. Referring to step one, we can add new column to the inventory table.

```
alter table inventory add columns(total double);
```

4. Running the select * command on inventory shows the new column has been added and all the entries are set to NULL. This command now executes

```
insert overwrite table inventory select item, units, unitcost,total
      from sales1;
```

5. There is no drop column command so we can use the following instead:

```
alter table inventory replace columns(product string, amount int,
      price double);
```

6. Use select to confirm that this is now the original inventory table

7. Drop the inventory table and recreate it with the total column added then load it from the sales1 table. Use the select * command to verify the table is loaded

```
drop table inventory;

create table inventory(
      product     string,
      amount      int,
      price       double,
      total       double)
stored as textfile;

insert into table inventory select item, units, unitcost, total
      from sales1;
```

8. Now drop the total column and verify with select * that the column total is dropped

```
alter table inventory replace columns(product string, amount int,
            price double);
```

9. Now add back the total column and a new column called rep.

```
alter table inventory add columns(total double, rep string);
```

10. Use the select * command to view the table contents. Why is there data in the total column but the rep column is all NULLs?

Dropping Tables

1. Use the drop table command to drop the inventory table.
2. Verify that it is no longer in the database (you should know how to do this now).
3. Verify that this has not altered the sales1 table.
4. Drop the sales1 table and any other tables that are in the salesdept database
5. Drop the salesdept database;