Lab 4: Hive Functions

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. Commands used in this lab are listed in the accompanying “commands.txt” file.

A lot of queries rely on the value of a column to be an atomic value when using relational operators in where clauses or other analysis. However, real world data is often not that convenient to use; for example, we may have an address column with an entire address as a single string but we want to select only those rows where the addresses are in specific city. We could transform the data to add a new column containing the city, but this is a lot of work if we are only going to be needed the city information once.

In this lab, you will use some of the Hive date functions to extract a year and month from a standard date as a single string. Then you will use a Hive string function to select some rows based on a partial match to a string

Data Setup

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

Creating the First Table

The schema for the SampleData.csv data is the same as for the previous lab:

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

1. For this lab, we will just use the default database.
2. Create an internal table sales4

```
create table sales4(
      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. Skip the first row of the table

```
alter table sales4 SET TBLPROPERTIES("skip.header.line.count"="1");
```

4. Load the data into the table

```
load data inpath'/user/maria_dev/data/SampleData.csv' into table sales4;
```

Date Functions

Use the select * function to look at the data. The first column is a date. You can use the various date functions to select various information from that field. In this case you will use the year and month functions to extract those values from the string. The advantage to using these functions over sub-string functions is that they can intelligently parse strings that represent dates and times

1. Start by listing all of the years that appear in the orderdates column.

```
select year(orderdate) from sales4;
```

2. However we can manage this better by suppressing the duplicate values

```
select distinct year(orderdate) from sales4;
```

3. We can also count the the number of distinct years in the data;

```
select count(distinct year(orderdate)) from sales4;
```

4. We can also use the date function in a where clause to select only those transactions that occur during May:

```
select * from sales4 where month(orderdate) = '5';
```

5. Or only the transactions that occurred March of 2022

```
select * from sales4
      where month(orderdate) = '3' and year(orderdate) = ‘2022’;
```

String Functions

The schema for the RecentGrads.csv data is:

```
major_code  string
mafor       string
rep         string
total       string
men         integer
women       integer
```

1. Create an internal table grads

```
create table grads(
      major_code  string,
      major       string,
      grads       int,
      men         int,
      women       int,)
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;
```

2. Skip the first row of the table

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

3. Load the data into the table

```
load data inpath'/user/maria_dev/data/RecentGrads.csv' into table grads;
```

4. Use the select * operation to view the data. Notice that all of the major descriptions are unique. However if you wanted to select only those majors that were Engineering majors, you have to look for values with the sub-string ‘ENGINEERING’ in it.

5. The insr(string, sub-string) returns the position that the sub-string occurs in string. If the sub-string is not in string, it returns a 0. You can use this to select only the rows where the major column contains the word ‘ENGINEERING’.

```
select major from grads where instr(major, 'ENGINEERING') > 0;
```

6. However, you might still get a mismatch if the case is wrong or there are trailing or leading spaces. Hive has a number of functions that we can use to fix up our data, like lower(string) that converts a string to lower case, to avoid these errors

```
select major from grads where instr(lower(major), 'engineering') > 0;
```

7. Also consider the output generated by this and guess what the string functions do.

```
select length(major), translate(reverse(major),'E','X') from grads;
```

Regular Expressions with rlike

This section assumes you are familiar with basic regular expressions. If not, there are a number of on-line tutorials you can refer to.

The rlike operator allows you to use pattern matching on strings. In this section, you will select various rows from the sales4 table looking for patterns in the name of the rep in the rep column.

To get a feel for the data, run the following queries to see what the rep names are:

```
select * from sales4;

select distinct rep from sales4;
```

The first query is to look far all the sales reps that have an 'a' in their name. Notice that we are using the lower function to prevent any case mismatches from capitalization.

```
select distinct rep from sales4 where lower(rep) rlike 'a';
```

The rerun this query with different patterns after the rlike select different rows

```
1. rlike '^a'        matches rep names that start with an 'a'

2. rlike 'n$'        matches rep names that end with a 'n'

3. rlike 'a.+r'      matches names that have an 'a' in them followed by a 'r'
                          with at least one letter separating them

4. rlike 'a.*r'      matches names that have an 'a' in them followed by a 'r'
    with at zero or more letters separating them
```

Experiment with other regular expressions.