Lab: Advanced Hive Queries

This lab explores some of the more advanced features of Hive work, including multi-table inserts, views, and windowing.

Objective: To understand how some of the more advanced features of Hive work, including multi-table inserts, views, and windowing.

File locations: ~/data

Successful outcome: You will have executed numerous Hive queries on customer order data.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>1. Create and Populate a Hive Table</h4>

From the command line, change directories to the data folder.

View the contents of the `orders.hive` file in that folder:

```
# more orders.hive
```

> *Note: you can get the file contents here and `shop.tsv` here.*

Notice it defines a Hive table named orders that has seven columns.

You can execute the contents of the file one after the other after logging into the Hive shell or execute the contents of `orders.hive` like this:

```
hive -f orders.hive
```

From the Hive shell, verify that the script worked by running the following commands:

```
hive> describe orders;
hive> select count(*) from orders;
```

Your orders table should contain 99,999 records.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>2. Analyze the Customer Data</h4>

Let’s run a few queries to see what this data looks like. Start by verifying that the `username` column actually looks like names:

```
hive> SELECT username FROM orders LIMIT 10;
```

You should see 10 first names.

The orders table contains orders placed by customers. Run the following query, that shows the 10 lowest-price orders:

```
hive> SELECT username, ordertotal FROM orders ORDER BY ordertotal LIMIT 10;
```

The smallest orders are each $10, as you can see from the output:

```
Chelsea     10
Samantha    10
Danielle    10
Kimberly    10
Tiffany     10
Megan       10
Maria       10
Megan       10
Melissa     10
Christina   10
```

Run the same query, but this time use descending order:

```
hive> SELECT username, ordertotal FROM orders ORDER BY ordertotal DESC LIMIT 10;
```

The output this time is the 10 highest-priced orders:

```
Brandon     612
Mark        612
Sean        612
Jordan      612
Anthony     612
Paul        611
Jonathan    611
Eric        611
Nathan      611
Jordan      610
```

Let’s find out if men or women spent more money:

```
hive> SELECT sum(ordertotal), gender FROM orders GROUP BY gender;
```

Based on the output, which gender has spent more money on purchases?

Answer: Men spent $9,919,847, and women spent $9,787,324.

The orderdate column is a string with the format yyyy-mm-dd. Use the year function to extract the various parts of the date. For example, run the following query, which computes the sum of all orders for each year:

```
hive> SELECT sum(ordertotal), year(order_date) FROM orders
 GROUP BY year(order_date);
```

The output should look like this. Verify, then quit the Hive shell:

```
4082780 2017
4404806 2014
4399886 2015
4248950 2016
2570769 2017


hive> quit;
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>3. Multi-File Insert</h4>

In this step, you will run two completely different queries, but in a single MapReduce job. The output of the queries
will be in two separate directories in HDFS. Start by using gedit (or editor of your choice) to create a new text file in your lab folder named `multifile.hive`.

Within the text file, enter the following query. Notice there is no semicolon between the two INSERT statements:

```
FROM ORDERS o
INSERT OVERWRITE DIRECTORY '2017_orders'
SELECT o.* WHERE year(order_date) = 2017
INSERT OVERWRITE DIRECTORY 'software'
SELECT o.* WHERE itemlist LIKE '%Software%';
```

Save your changes to `multifile.hive`.

Run the query from the command line:

```
# hive -f multifile.hive
```

The above query executes in a single MapReduce job. Even more interesting, it only requires a map phase.

Why did this job not require a reduce phase?

Answer: Because the query only does a SELECT *, no reduce phase was needed.

Verify that the two queries executed successfully by viewing the folders in HDFS:

```
# hdfs dfs -ls
```

You should see two new folders: `2017_orders` and `software`.

View the output files in these two folders. Verify that the `2017_orders` directory contains orders from only the year 2017,
and verify that the `software` directory contains only orders that included ‘Software.’

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>4. Define a View</h4>

Start the Hive shell. Define a view named `2016_orders` that contains the orderid, order_date, username, and itemlist columns of the `orders` table where the order_date was in the year 2016.

Solution: The `2016_orders` view:

```
hive> CREATE VIEW 2016_orders AS SELECT orderid, order_date, username, itemlist FROM orders
WHERE year(order_date) = '2016';
```

Run the show tables command:

```
hive> show tables;
```

You should see `2016_orders` in the list of tables.

To verify your view is defined correctly, run the following query:

```
hive> SELECT COUNT(*) FROM 2016_orders;
```

The `2016_orders` view should contain around 21,544 records.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>5. Find the Maximum Order of each Customer</h4>

Suppose you want to find the maximum order of each customer. This can be done easily enough with the following Hive query.

Run this query now:

```
hive> SELECT max(ordertotal), userid FROM orders GROUP BY userid;
```

How many different customers are in the orders table?

Answer: There are 100 unique customers in the orders table.

Suppose you want to add the itemlist column to the previous query. Try adding it to the SELECT clause by the following method and see what happens:

```
hive> SELECT max(ordertotal), userid, itemlist FROM orders
 GROUP BY userid;
```

Notice this query is not valid because itemlist is not in the GROUP BY key.

We can join the result set of the max-total query with the orders table to add the itemlist to our result. Start by defining a view named max_ordertotal for the maximum order of each customer:

```
hive> CREATE VIEW max_ordertotal AS
SELECT max(ordertotal) AS maxtotal, userid FROM orders
GROUP BY userid;
```

Now join the orders table with your max_ordertotal view:

```
hive> SELECT ordertotal, orders.userid, itemlist FROM orde
rs
JOIN max_ordertotal
ON max_ordertotal.userid = orders.userid
AND
max_ordertotal.maxtotal = orders.ordertotal
ORDER BY orders.userid;
```

The end of your output should look like:

```
600 98  Grill,Freezer,Bedding,Headphones,DVD,Table,Grill,Software,Dishwasher,DVD,Microwave,Adapter
600 99  Washer,Cookware,Vacuum,Freezer,2-Way Radio,Bicycle,Washer & Dryer,Coffee Maker,Refrigerator,DVD,Boots,DVD
600 100 Bicycle,Washer,DVD,Wrench Set,Sweater,2-Way Radio,Pants,Freezer,Blankets,Grill,Adapter,pillows
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>6. Fixing the GROUP BY key Error</h4>

Let’s compute the sum of all of the orders of all of the customers. Start by entering the following query:

```
hive> SELECT sum(ordertotal), userid FROM orders GROUP BY userid;
```

Notice that the output is the sum of all orders, but displaying just the userid is not very exciting.

Try to add the username column to the SELECT clause in the following manner and see what happens:

```
hive> SELECT sum(ordertotal), userid, username FROM orders
```

```
  GROUP BY userid;
```

This generates the infamous “Expression not in GROUP BY key” error, because the username column is not being aggregated but the ordertotal is.

An easy fix is to aggregate the username values using the collect_set function, but output only one of them:

```
hive> SELECT sum(ordertotal), userid, collect_set(username
)[0] FROM orders GROUP BY userid;
```

You should get the same output as before, but this time the username is included.

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>7. Using the OVER Clause</h4>

Now let’s compute the sum of all orders for each customer, but this time use the OVER clause to not group the output and to also display the itemlist column:

```
hive> SELECT userid, itemlist, sum(ordertotal) OVER (PARTI
TION BY userid) FROM orders;
```

> *NOTE: the output contains every order, along with the items they purchased and the sum of all of the orders ever placed from that particular customer.*

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>8. Using the Window Functions</h4>

It is not difficult to compute the sum of all orders for each day using the GROUP BY clause:

```
hive> select order_date, sum(ordertotal) FROM orders GROUP
 BY order_date;
```

Run the query above and the tail of the output should look like:

```
2017-07-28  18362
2017-07-29  3233
2017-07-30  4468
2017-07-31  4714
```

Suppose you want to compute the sum for each day that includes each order. This can be done using a window that sums all previous orders along with the current row:

```
hive> SELECT order_date, sum(ordertotal) OVER
(PARTITION BY order_date ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW)
FROM orders;
```

To verify that it worked, your tail of your output should look like:

```
2017-07-31  3163
2017-07-31  3415
2017-07-31  3607
2017-07-31  4146
2017-07-31  4470
2017-07-31  4610
2017-07-31  4714
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>9. Using the Hive Analytics Functions</h4>

Run the following query, which displays the rank of the ordertotal by day:

```
hive> SELECT order_date, ordertotal, rank() OVER
(PARTITION BY order_date ORDER BY ordertotal) FROM orders;
```

To verify it worked, the output of July 31, 2017, should look like:

```
2017-07-31  48  1
2017-07-31  104 2
2017-07-31  119 3
2017-07-31  130 4
2017-07-31  133 5
2017-07-31  135 6
2017-07-31  140 7
2017-07-31  147 8
2017-07-31  156 9
2017-07-31  192 10
2017-07-31  192 10
2017-07-31  196 12
2017-07-31  240 13
2017-07-31  252 14
2017-07-31  296 15
2017-07-31  324 16
2017-07-31  343 17
2017-07-31  500 18
2017-07-31  528 19
2017-07-31  539 20
```

As a challenge, see if you can run a query similar to the previous one except compute the rank over months instead of each day.

Solution: The rank query by month:

```
SELECT substr(order_date,0,7), ordertotal, rank() OVER
(PARTITION BY substr(order_date,0,7) ORDER BY ordertotal)
FROM orders;
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo” />
<h4>10. Histograms</h4>

Run the following Hive query, which uses the histogram_numeric function to compute 20 (x,y) pairs of the frequency distribution of the total order amount from customers who purchased a microwave (using the orders table):

```
hive> SELECT explode(histogram_numeric(ordertotal,20))
AS x FROM orders
WHERE itemlist LIKE "%Microwave%";
```

The output should look like the following:

```
{"x":14.333333333333332,"y":3.0}
{"x":33.87755102040816,"y":441.0}
{"x":62.52577319587637,"y":679.0}
{"x":89.37823834196874,"y":965.0}
{"x":115.1242236024843,"y":1127.0}
```

```
{"x":142.6468885672939,"y":1382.0}
{"x":174.07664233576656,"y":1370.0}
{"x":208.06909090909105,"y":1375.0}
{"x":242.55486381322928,"y":1285.0}
{"x":275.8625954198475,"y":1048.0}
{"x":304.71100917431284,"y":872.0}
{"x":333.1514423076924,"y":832.0}
{"x":363.7630208333335,"y":768.0}
{"x":397.51587301587364,"y":756.0}
{"x":430.9072847682117,"y":604.0}
{"x":461.68715083798895,"y":537.0}
{"x":494.1598360655734,"y":488.0}
{"x":528.5816326530613,"y":294.0}
{"x":555.5166666666672,"y":180.0}
{"x":588.7979797979801,"y":198.0}
```

Write a similar Hive query that computes 10 frequency-distribution pairs for the ordertotal from the orders table
where ordertotal is greater than $200.

```
SELECT explode(histogram_numeric(ordertotal,10)) AS x FROM
 orders
WHERE ordertotal > 200;


{"x":218.8195174551819,"y":7419.0}
{"x":254.10237580993478,"y":6945.0}
{"x":293.4231618807192,"y":6338.0}
```

```
{"x":334.57302573203015,"y":5635.0}
{"x":379.79714934930786,"y":4841.0}
{"x":428.1165628891644,"y":4015.0}
{"x":473.1484734420741,"y":2391.0}
{"x":511.2576946288467,"y":1657.0}
{"x":549.0106899902812,"y":1029.0}
{"x":589.0761194029857,"y":670.0}
```

Result

You should now be comfortable running Hive queries and using some of the more advanced features of Hive, like views and the window functions.