Recommending Rail Tunnel Route in the US with SQL

The purpose of this project is to use SQL statement in order to find the best route for a fictional transportation company that wants to build a rail tunnel that connects two airports based on several requirements or criteria. The data being used for this project are saved in Hadoop Distributed File System (HDFS). In order to query the data, Apache Hive or Impala will be used.

Next after we find out which route will be the best for rail tunneling, let's say that the company starts the tunneling process. Each day, new data is continuously being generated and finally, the final data are stored in the cloud storage, which in this project is AWS S3. The second purpose of this project is to fetch these final tunneling data from cloud storage and migrate them into HDFS clusters.

First Case: Rail Tunnel Route Recommendation

There is a company that plans to disrupt the airline industry by building an underground high-speed passenger rail tunnel. The company needs help to decide which two major United States airports this tunnel should connect. The distance between the airports must be within a specified range, and the airports must have a large volume of air travelers flying between them in both directions. The company believes that these air travelers can be persuaded to switch to high-speed rail because of frustratingly long flight delays

.

Requirements

The job is to recommend which pair of United States airports should be connected with a high-speed passenger rail tunnel. The company has given the following strict requirements:

These two airports must:

1. Be between 300 and 400 miles apart.
2. Average at least 5,000 (five thousand) flights per year between them, in each direction.
3. Among the pairs of airports that meet these requirements, you must identify the one pair that has the largest total number of seats on the planes that flew between them.

4. The company is also interested to know the average arrival delay for flights between these two airports, because they believe that routes with a history of delayed arrivals will make it easier to persuade air travelers to switch to high-speed rail.

SQL Query

In order to fulfill all of the requirements above, we need to create SQL statement of the data that are saved in the HDFS, particularly in **fly** database (The database overview and the schema of this database is given in a separate file). The following SQL statement will do the trick for this problem.

```
SELECT round(sum(planes.seats)/10) as avg_seats,round(sum(seats)) as
    tot_seats, round(count(flight)/10) as avg_flight,
avg(distance) as avg_dist, round(avg(arr_delay),2) as avg_delay,origin,
    dest
FROM flights LEFT JOIN planes
ON flights.tailnum = planes.tailnum
WHERE distance BETWEEN 300 AND 400
GROUP BY origin, dest
HAVING avg_flight > 5000
ORDER BY tot_seats DESC NULLS LAST
LIMIT 10;
```

Table 0.1: SQL output from table flights in **fly** database

avg_seats	tot_seats	avg_flight	avg_dist	avg_delay	origin	dest
1996597	19965970	14712	337	10.32	SFO	LAX
1981059	19810585	14540	337	13.76	LAX	SFO
1219235	12192349	8662	370	5.82	PHX	LAX
1210173	12101731	8650	370	5.95	LAX	PHX
1067278	10672784	6200	304	4.89	PHX	SAN
1060204	10602041	6216	304	3.77	SAN	PHX
920919	9209185	8012	390.6225631	4.22	SLC	DEN
893437	8934374	7667	390.6222089	6.24	DEN	SLC
867688	8676883	8484	399	1.35	BOS	DCA
864009	8640091	8493	399	4.03	DCA	BOS

From the table above, we can see and conclude that the recommended tunneling route, with considering all of the criteria and requirements that have been defined before, would be in between San Francisco Airport and Los Angeles Airports. Both

of the airports have the most frequent flights between them compared to other airports and the distance between them is also in between 300-400 apart. These two airports also have the most passenger traveling between them. Hence, the tunnel connecting San Francisco and Los Angeles would be ideal.

Second Case: Migrating Tunnel Boring Data from Cloud to HDFS

Based on the analysis and on other factors, construction has begun on a tunnel connecting San Francisco and Los Angeles. The tunnel will be dug over a period of ten years. It will be dug in three different sections by three tunnel boring machines (TBMs) named Bertha II, Shai-Hulud, and Diggy McDigface.

Each of these TBMs will generate a large volume of data as it operates. There are three TBM files containing the tunnel boring machine data. They are delimited text files, each containing tens of thousands of lines. They are stored in Amazon S3 in sub-directories under a directory named tbm_sf_la in the S3 bucket named training-coursera2.

Challenges

These are the things that these three files have in common:

1. Each file contains eight columns representing the same eight fields.
2. The data types of the eight columns are the same in all three files.
3. The rows of the table represent hourly time intervals.

These are the differences between these three files:

1. They use different delimiters.
2. One of the files uses the string 999999 to represent missing values.
3. One of the files has a header line.

The main goal in this section is to fetch these three different data from cloud services and then migrate them into HDFS. After migrating all of the data, we need to store these three data into one table called tbm_sf_la.

Checking the Files Inside the Bucket in AWS S3

Before we start with the SQL statement to migrate the data, let's check what kinds of data that have been stored in AWS S3. In order to do this, let's write a command line.

```
[training@localhost ~]$ hdfs dfs -ls s3a://training-coursera2/tbm_sf_la/
```

```
drwxrwxrwx  - training training       0 2020-05-08 04:02
   s3a://training-coursera2/tbm_sf_la/central
drwxrwxrwx  - training training       0 2020-05-08 04:02
   s3a://training-coursera2/tbm_sf_la/north
drwxrwxrwx  - training training       0 2020-05-08 04:02
   s3a://training-coursera2/tbm_sf_la/south
```

There are three different sub-directories in the bucket named training-coursera2, which are central, north, and south. Next, let's find out what is inside these three directories.

```
[training@localhost ~]$ hdfs dfs -cat
   s3a://training-coursera2/tbm_sf_la/central/hourly_central.csv | head
```

```
tbm,year,month,day,hour,dist,lon,lat
Shai-Hulud,2020,01,02,09,0.00,-121.345467,37.599819
Shai-Hulud,2020,01,02,10,4.90,999999,999999
Shai-Hulud,2020,01,02,11,9.79,999999,999999
Shai-Hulud,2020,01,02,12,14.69,999999,999999
Shai-Hulud,2020,01,02,13,19.59,999999,999999
```

As we can see, there is a csv file called hourly_central.csv inside of the sub-directory central. This csv file has a column header, it is comma delimited, and it treats NULL values as 999999. We need to convert this 999999 back to NULL after we migrate it into HDFS.

Next, let's find out what is inside north sub-directory.

```
[training@localhost ~]$ hdfs dfs -cat
   s3a://training-coursera2/tbm_sf_la/north/hourly_north.csv | head
```

```
Bertha II,2020,01,02,09,0.00,-121.345947,37.600201
Bertha II,2020,01,02,10,5.00,\N,\N
```

```
Bertha II,2020,01,02,11,10.00,\N,\N
Bertha II,2020,01,02,12,15.00,\N,\N
Bertha II,2020,01,02,13,20.00,-121.346107,37.600319
```

There is also a csv file called hourly_north.csv inside of the sub-directory central. This csv file doesn't have a column header, it is comma delimited, and it treats NULL values as it is.

Next, let's find out what is inside south sub-directory.

```
[training@localhost ~]$ hdfs dfs -cat
   s3a://training-coursera2/tbm_sf_la/south/hourly_south.tsv | head
```

```
Diggy McDigface 2020 01 02 09 0.00 -118.933868 34.949688
Diggy McDigface 2020 01 02 10 1.16 \N \N
Diggy McDigface 2020 01 02 11 2.32 \N \N
Diggy McDigface 2020 01 02 12 3.49 \N \N
Diggy McDigface 2020 01 02 13 4.65 \N \N
```

Different from other files, the file called hourly_south.tsv in the south sub-directory has .tsv extension on it, which means it is tab delimited file, as we can see from the result above. So, we need to treat these three files differently in order to merge them together into one coherent table at the end.

Creating Three Tables in HDFS Dig Database Based on Three Files in S3

Next, we want to fetch all of the data from the cloud and migrate them into clusters. The database in the HDFS to store the tables for these data is called dig. However, we need to create a separate table for each of these three files because they have different formats.

First, let's create a table for hourly_central.csv file in HDFS and put the content of the data to the newly created table. In order to skip the column header, **skip.header.line.count** syntax from **TBLPROPERTIES** will be used. In order to convert 999999 into NULL values, **serialization.null.format** from **TBLPROPERTIES** will be used.

```
 CREATE EXTERNAL TABLE dig.hourly_central
(tbm STRING, year SMALLINT, month TINYINT, day TINYINT, hour TINYINT,
    dist  DECIMAL(8,2), lon FLOAT, lat FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3a://training-coursera2/tbm_sf_la/central/'
TBLPROPERTIES ('serialization.null.format' =
    '999999','skip.header.line.count' = '1');
```

```
SELECT * FROM dig.hourly_central LIMIT 5;
```

Table 0.2: Output of table dig.hourly_central

tbm	year	month	day	hour	dist	lon	lat
Shai-Hulud	2020	1	2	9	0.00	-121.3454666	37.59981918
Shai-Hulud	2020	1	2	10	4.90	NULL	NULL
Shai-Hulud	2020	1	2	11	9.79	NULL	NULL
Shai-Hulud	2020	1	2	12	14.69	NULL	NULL
Shai-Hulud	2020	1	2	13	19.59	NULL	NULL

And there we have it. The first tunnel boring machine data in a table and stored in dig database in clusters. Next, let's do the same with hourly_north.csv. However, since this file doesn't have any column header and the NULL value is already as it is, the syntax will be simpler.

```
CREATE EXTERNAL TABLE dig.hourly_north
(tbm STRING, year SMALLINT, month TINYINT, day TINYINT, hour TINYINT,
    dist  DECIMAL(8,2), lon FLOAT, lat FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3a://training-coursera2/tbm_sf_la/north/';
```

```
SELECT * FROM dig.hourly_north LIMIT 5;
```

Table 0.3: Output of table dig.hourly_north

tbm	year	month	day	hour	dist	lon	lat
Bertha II	2020	1	2	9	0.00	-121.3459473	37.60020065
Bertha II	2020	1	2	10	5.00	NULL	NULL
Bertha II	2020	1	2	11	10.00	NULL	NULL
Bertha II	2020	1	2	12	15.00	NULL	NULL
Bertha II	2020	1	2	13	20.00	-121.3461075	37.60031891

And the output of the table in dig database is as we expected. Next, let's migrate the hourly_south.tsv file from cloud. However, since this file has .tsv extension, then we need to adjust the syntax **ROW FORMAT DELIMITED FIELDS TERMINATED BY** from ',' to '\t'

```
CREATE EXTERNAL TABLE dig.hourly_south
(tbm STRING, year SMALLINT, month TINYINT, day TINYINT, hour TINYINT,
   dist   DECIMAL(8,2), lon FLOAT, lat FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3a://training-coursera2/tbm_sf_la/south/'
```

```
SELECT * FROM dig.hourly_south LIMIT 5;
```

Table 0.4: Output of the table hourly_south

tbm	year	month	day	hour	dist	lon	lat
Diggy McDigface	2020	1	2	9	0.00	-118.9338684	34.94968796
Diggy McDigface	2020	1	2	10	1.16	NULL	NULL
Diggy McDigface	2020	1	2	11	2.32	NULL	NULL
Diggy McDigface	2020	1	2	12	3.49	NULL	NULL
Diggy McDigface	2020	1	2	13	4.65	NULL	NULL

.

.

Creating Final Table tbm_sf_la and Merge Three Tables Together

Next, we want to merge all of these three tables together into one table called tbm_sf_la. In order to do this, first create an empty table in dig database, and then merge all of the data in three different tables above together using **UNION ALL** syntax.

```
CREATE EXTERNAL TABLE dig.tbm_sf_la
  (tbm STRING, year SMALLINT, month TINYINT, day TINYINT, hour TINYINT,
     dist  DECIMAL(8,2), lon FLOAT, lat FLOAT)
```

```
INSERT INTO dig.tbm_sf_la
SELECT * FROM dig.hourly_central
UNION ALL
SELECT * FROM dig.hourly_north
UNION ALL
SELECT * FROM dig.hourly_south;
```

And then query the table tbm_sf_la.

```
SELECT tbm, COUNT(*) AS num_rows FROM dig.tbm_sf_la GROUP BY tbm ORDER
   BY tbm;
```

Table 0.5: Output of table tbm_sf_la

tbm	num_rows
Bertha II	91619
Diggy McDigface	93163
Shai-Hulud	94237

And there we have the final result of a table stored in HDFS consisting the merge of three different files with three different formats using SQL statement.