Getting to grips with Hive

Week 2

If you want to really do some serious playing with Hadoop you probably need this set up on your own PC or Laptop. Instructions for this are given in a separate document in Blackboard.

Before you read any more of this tutorial, begin by copying the VM either from the VMStore, or if you have your own from last week, from your pen-drive. Once it is on your D: drive, start the VM in the way we did last week. Whilst the copying/starting is going on, you can be reading the next bit!

Expect this tutorial, together with the Pig tutorial, to take longer than the time allotted, unless you are very clever! So get as far as you can and then complete in your own time, emailing any questions to p.lake@shu.ac.uk. We suggest you swap from Hive to Pig after an hour or so, just so you can be trying it out whilst tutors are around.

NOTE: If you are running this on your own PC and it has 8Gb RAM, be aware that you will need to amend the RAM for the VM to 4Gb (See separate notes on using on your own PC). It will then run very slowly, as I can attest! The sandbox will not work on a PC with less than 8Gb RAM. (see https://community.hortonworks.com/questions/35000/hdp-sandbox-startup-too-long-on-virtualbox.html)

Using Hive to ingest data

NOTE: This tutorial does not explain the SQL used in great detail. If you are unsure about SQL we suggest you look at one of the many online tutorials available. The W3Schools tutorials are a good example: http://www.w3schools.com/sql/ .

Hive lets you project structure onto Hadoop data, allowing it to be queried with an SQL-like language called HiveQL. Metadata about the tables stored help present something like a relational view of data, which can make accessing Hadoop data much easier for those who already have SQL skills. Just like a RDBMS, the metadata store presents data in tables which are themselves stored in databases.

This relational-like table abstraction means that users do not need to worry about what the source format of the data is, nor where it is stored. In terms of incoming data sources Hive supports CSV (see http://en.wikipedia.org/wiki/Comma-separated_values) and JSON files (see http://en.wikipedia.org/wiki/JSON), amongst other formats, and we will be using examples of both in these tutorials.

We shall now download and use a simple CSV from the http://ourairports.com/data/ website which has freely usable datasets available. The one we will use first contains information about the airports of the world. It tells us, amongst other things, the latitude and longitude, and elevation of each airport. It can be found here: http://ourairports.com/data/airports.csv. Download the CSV to your host PC. If you want to look at its contents you should use a text editor like Textpad (see figure 1), rather than double-clicking on it, since the clicking may well open an unwanted spreadsheet.

```
"id","ident","type","name","latitude_deg","longitude_deg","elevation_ft","continent","iso_country","iso_region","municipality","scheduled_service","gps_code","iata_code"
6523,"00A","heliport","Total Rf Heliport",40.07080078125,-74.9336013793945,11,"NA","US","US-PA","Bensalem","no","00A",,"00A",,,
6524,"00AK","small_airport","Lowell Field",59.94919968,-151.695999146,450,"NA","US","US-AK","Anchor Point","no","00AK",,"00AK",,,
6525,"00AL","small_airport","Epps Airpark",34.8647994995117,-86.7703018188477,820,"NA","US","US-AL","Harvest","no","00AL",,"00AL",,,
6526,"00AR","heliport","Newport Hospital & Clinic Heliport",35.608699798584,-91.2548980712891,237,"NA","US","US-AR","Newport","no","00AR",,"00AR",,,
6527,"00AZ","small_airport","Cordes Airport",34.3055992126465,-112.165000915527,3810,"NA","US","US-AZ","Cordes","no","00AZ",,"00AZ",,,
6528,"00CA","small_airport","Goldstone /Gts/ Airport",35.3504981995,-116.888000488,3038,"NA","US","US-CA","Barstow","no","00CA",,"00CA",,,
6529,"00CO","small_airport","Cass Field",40.622200012207,-104.34400177002,4830,"NA","US","US-CO","Briggsdale","no","00CO",,"00CO",,,
6531,"00FA","small_airport","Grass Patch Airport",28.64550018310...,-82.21900177001...,53,"NA","US","US-FL","Bushnell","no","00FA",,"00FA"
```

Figure 1

You will note that the first row is a header which describes what each column contains. The comma is used to separate the values for each column.

We need to upload the CSV file. Because the CSV has a header, we can allow Hive to use that to create column names automatically. Once you have downloaded the CSV to your host PC, start a Hive View session using the Ambari menu (see figure 2).

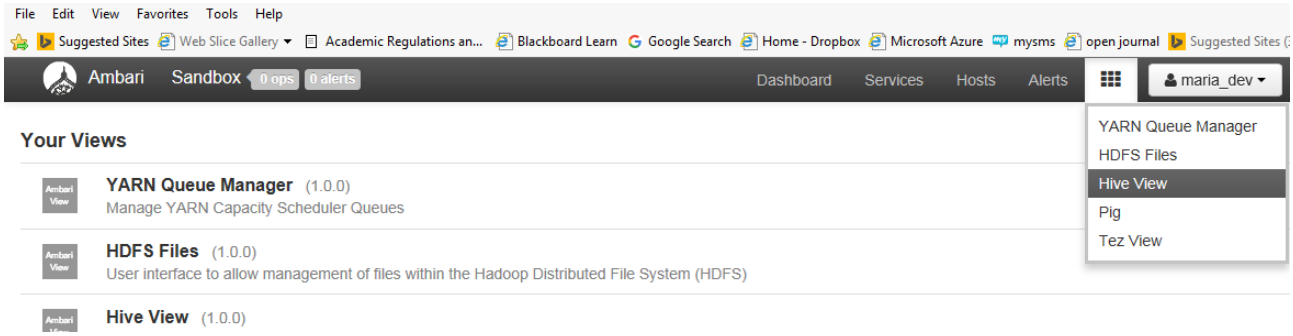

Figure 2

The screen that now appears is your Hive working area. There is a worksheet for SQL queries that we will shortly be using, but first we will upload the airports CSV into Hive, asking it to create a table for us using the header. Click on the Upload Table button (figure 3).

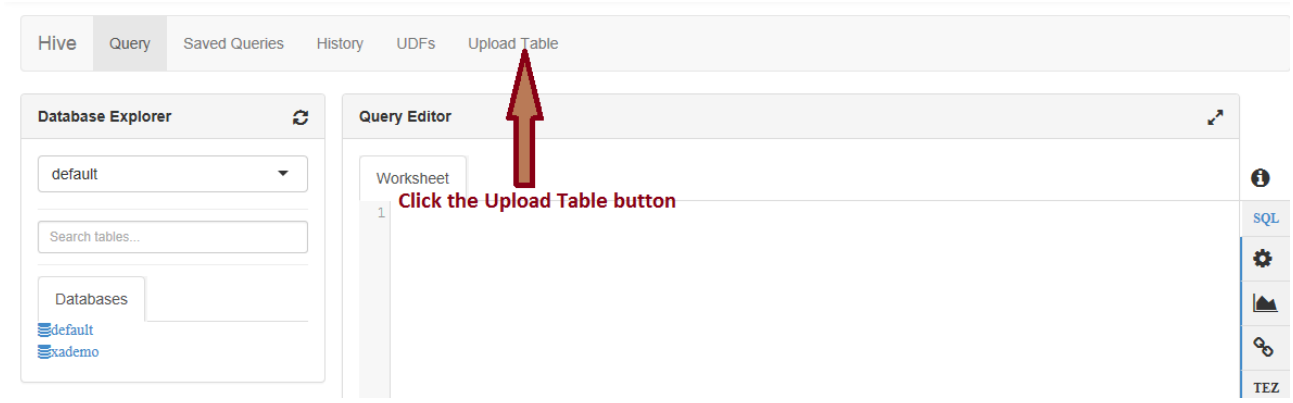

Figure 3

At the next screen you need to click on the Browse button and go and select your CSV file. Your screen should now look something like figure 4. For the time being we will be using the *default* database.

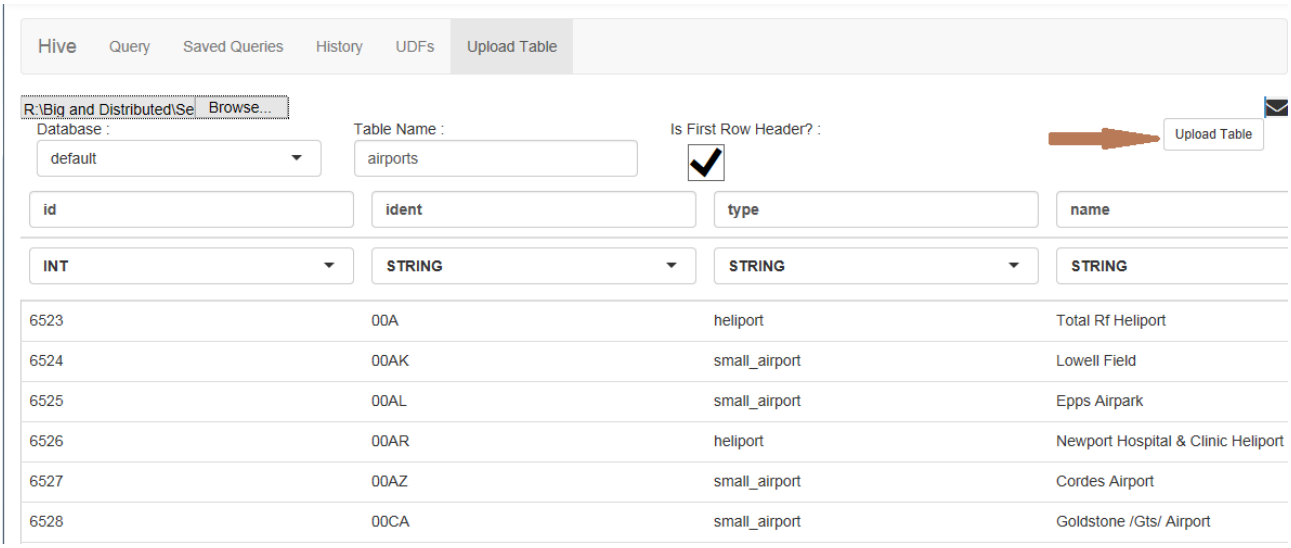

Figure 4

This process of creating the table structure has been easy because we had a header. Often we will need to create structure for ourselves.

Now we can click on the Upload Table on the right of this form (see brown arrow on figure 4). You should then get the *Uploaded Successfully* message. To prove this has worked, return to the query tab and type this into your worksheet:

select * from airports ;

Then click Execute and you should end up with something like figure 5.

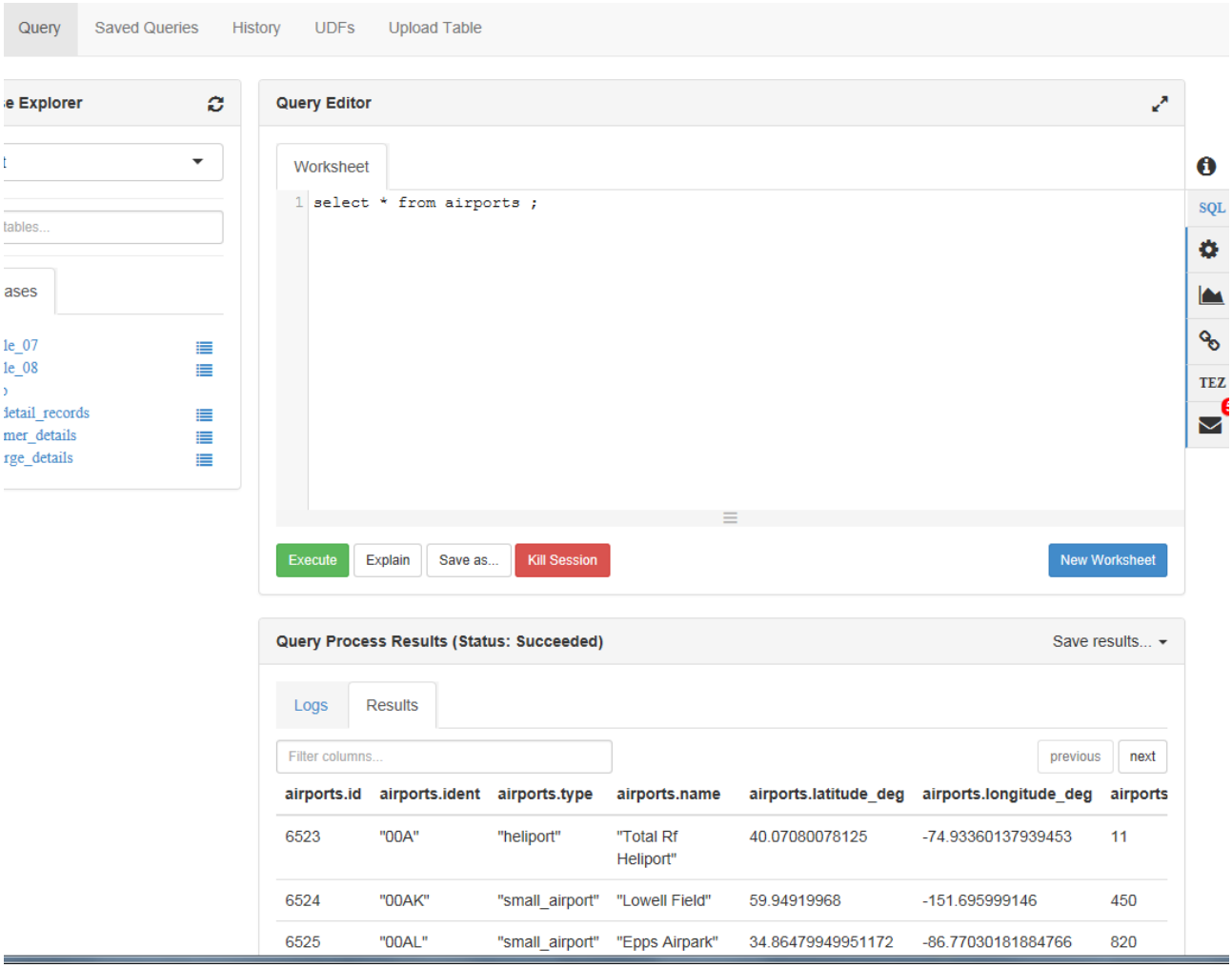

Figure 5

Let us just be clear what has happened here. We have captured some data in CSV format from the internet. We stored that CSV on our Host PC. We then logged into our Hadoop VM, and used a Hive tool to read that CSV. It used the header to create a table structure and then brought the data into a table. You have now stored your first data into the Hive repository running on Hadoop!

Data Cleansing

That process was relatively straightforward. However, it is not unusual for us to have to manipulate the incoming data in some way to make it more suitable for our needs. This may be some significant task, such as summarising, or filtering. In this case, if you look again at the output in Figure 5, you will see that our text data has quotes around it. It means that, in order to use this data, we would have to write, for example:

select * from airports where type = '"heliport"' ;

Instead of the more natural feeling:

select * from airports where type = 'heliport' ;

This may not seem like a big deal to you now, but your users will certainly not thank you for making life a little more difficult for them!

In this case, we need simply to remove the quotes. Take a copy of the original data before messing around with it; save this as something like airports2.csv. Open the airports2.csv using a text editor, such as Texpad (Textpad is available from the start menu in the labs). Use the global replace, under the Search menu, to remove the quotes (see Figure 6). Of course we have to be very careful with our data. This could be a very dangerous thing to do. We examine moving data in more detail in later weeks. For the time being, just take our word for it that this does not make this particular data any less valid.

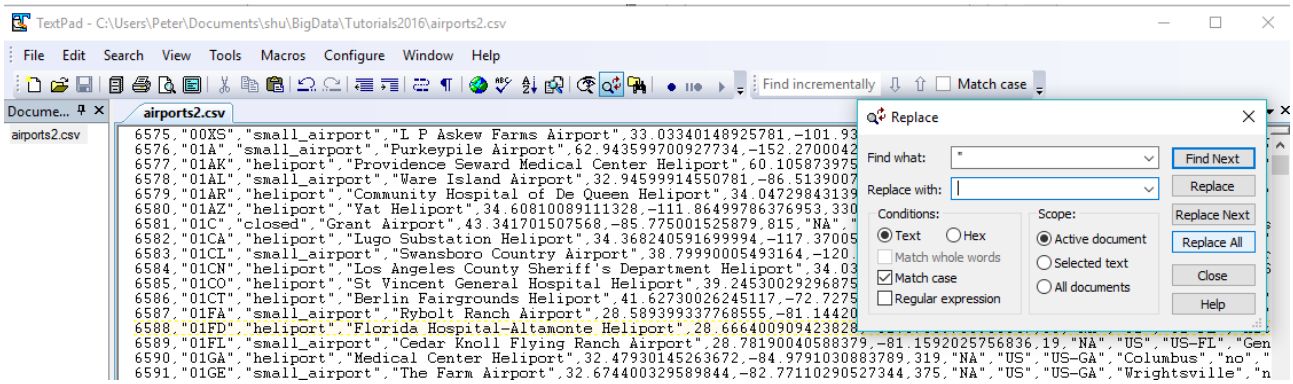

Figure 6

Next, we need to remove the bad table we imported first. Return to the worksheet area and execute:

drop table airports ;

If you are new to SQL, you could refer to this to see what ***drop*** means:
http://www.orafaq.com/faq/difference_between_truncate_delete_and_drop_commands

Now we can recreate the airports table, but point to the file which has the quotes removed.

Using Hive to query the data

Return to the worksheet tab of the Hive View. Now we have the data available in Hive we begin to answer questions like: How many Heliports are there in India?

If you already know SQL you will recognise that in the first step (Figure 7), we are starting to ask the opening question. Copy the following code into the query editor and press the *Execute* button:

```
select * from airports
where iso_country = 'IN'
AND type = 'heliport' ;
```

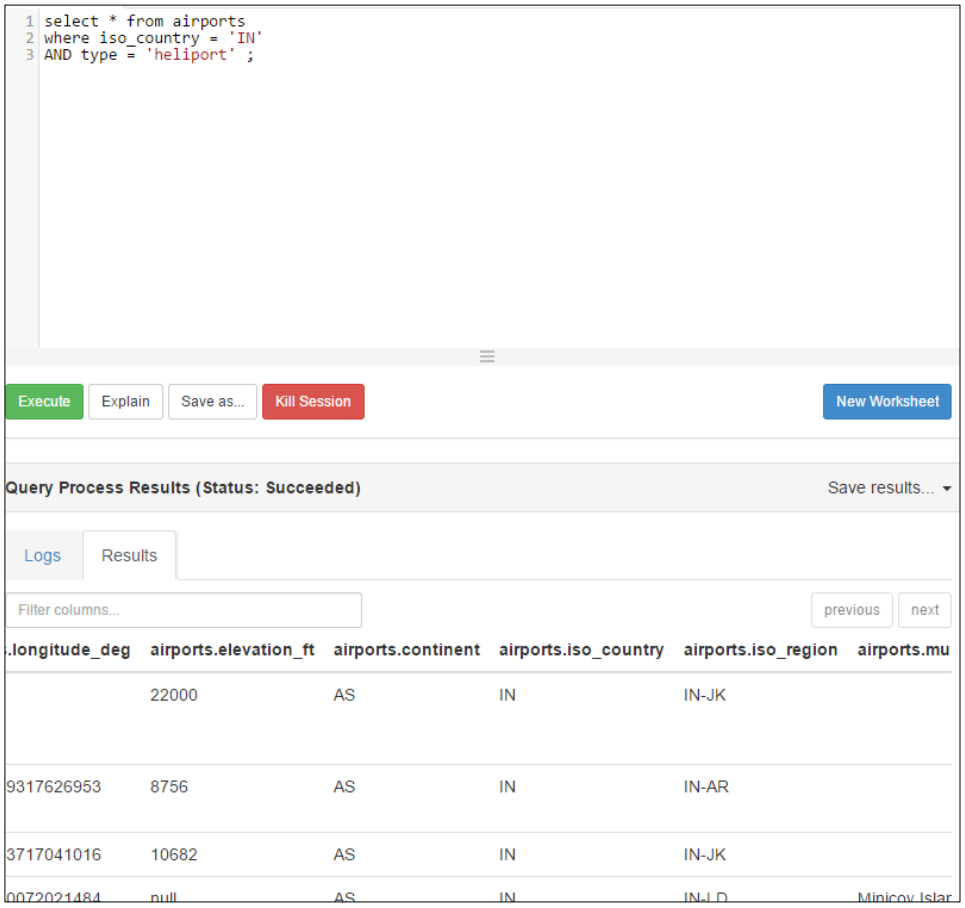

Figure 7

Click on the **Save As** button to keep this query for use in the future. You can see your saved queries by clicking the Saved Queries tab.

Actually what we have above is merely a list of heliports in India. Since the question posed was '*How many Heliports are there in India*?' what we need is a count. Now click on the New Worksheet button and copy in this query, and execute it:

```
select count(*) as Number from airports
where iso_country = 'IN'
AND type = 'heliport' ;
```

When you execute this you will see it will take a lot longer than a simple list takes. This is because there is some analysis to be done and the HiveQL needs to be translated into a MapReduce job, which is what we see in the logs section as we wait for the output. Running this on my 8Gb server at home is a lot slower than running it in the labs – these queries are RAM hungry. So long as the Process Results message says ***Running***, you are alright, but you might have to make a cup of coffee between exercises later on!

When you get more comfortable with Ambari you can monitor the sort of usage your query is putting your system to by opening a separate browser tab and just looking at the Ambari home page, but for the moment, be patient, you will eventually get a result. At the time of writing the answer is 40 (see Figure 8).

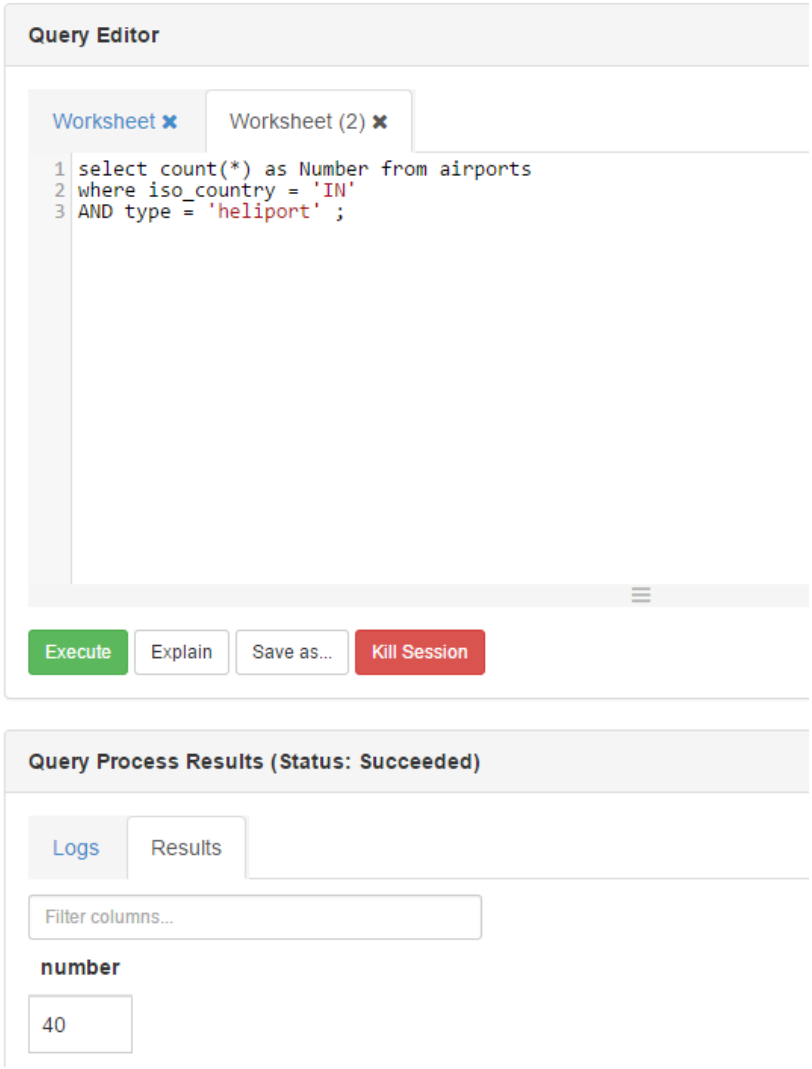

Figure 8

For interest the iso_country referred to here is an International Organization for Standardization (ISO) code for each country. The airport identifier referred to is: http://en.wikipedia.org/wiki/International_Civil_Aviation_Organization_airport_code

If you have SQL skills you could doubtless think of countless other queries you could run against this data. Indeed you should try some for yourself. Here is one more worked example before we move on: Which large airport in India is at the highest elevation?

From our first query we know that we used the WHERE clause to specify the condition that rows should contain the value “IN” in the iso_country column. And we also have used the AND before to combine conditions. We could therefore start with:

```
select name, elevation_ft from airports
where iso_country = "IN"
AND type = "large_airport" ;
```

We could then use the ORDER BY clause to present the data returned above in sorted order:

```
select name, elevation_ft from airports
where iso_country = "IN"
AND type = "large_airport"
ORDER BY elevation_ft ;
```

What answer do you get?

However, what we have now is just a list of airports sorted by elevation. What we actually asked was; which was THE highest airport, so we want only one row returned. Those used to writing SQL might already have thought of a solution to this using what is called a nested sub-query, in which the right hand side of the WHERE clause contains the result of another query; in this case the value of the maximum elevation_ft. Your ideas might look something like this:

```
select name, elevation_ft from airports
where elevation_ft =
(select max(elevation_ft) from airports
 where iso_country = "IN"
 AND type = "large_airport"  ) ;
```

By all means try this in HiveQL, but I am sorry to say you will get an error! This highlights something that the SQL aficionados need to remember: HiveQL is not quite the same as ISO Standard SQL and some of the functionally available in standard RDBMS query engines are just not available in Hive. And we have discovered one here!

So how do we find the answer we are looking for? As is so often the case with software, there is more than one way of tackling the problem. This may not be as elegant as a nested query, nor quite so readable, but try the following:

```
SELECT name, elevation_ft
FROM airports
JOIN
(  SELECT Max(elevation_ft) AS maxelevation
   FROM airports
   WHERE iso_country = "IN"
   AND type = "large_airport"
) maxdata
ON airports.elevation_ft = maxdata.maxelevation
WHERE iso_country = "IN"
AND type = "large_airport" ;
```

In this query we have resorted to using a JOIN which, as we will see shortly, is more often used to combine rows from two or more tables, based on a common column. In this case we are joining the airports table to a subset of itself (that which is in brackets) which only contains the rows which have the maximum elevation for an Indian large airport. We cannot assume there will only be one airport in the world which is at that altitude, so the final WHERE restricts the newly formed dataset to only those large airports whic
h are in India.

The message from this is be prepared to rethink! Who knows if HiveQL will ever include all the functionality of one of the commercial RDBMS systems, but it is probable that you can always find a way of getting the answer we want.

We will now add a second table to our Hive metastore to allow us to do some joins. One of the things Data Scientists can do is use data from a variety of sources, both from within an organisation and without, and both structured and unstructured. We are going to add value to our airports table by bringing in some data that was made available by the Guardian (http://www.theguardian.com/news/datablog/2012/may/04/world-top-100-airports#data) which published the top 100 airports by passengers. The data they used is available in Blackboard in the T100data.csv file. You will note there is no header in this file. We will be demonstrating another technique for getting data into Hive. You should first use the HDFS file navigator to upload the T100data.csv file and put it into your tutorials directory within HDFS, as you did with Joyce.txt last week.

Our next step will be to create a table to accept the data. We do this in the Hive Worksheet area with a straightforward SQL command as below:

```
Create Table T100 (
rank INT,
reg STRING,
code STRING,
airport STRING,
airport_location STRING ,
passengers_2011 INT,
passengers_2010 INT,
change DOUBLE )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ","
ESCAPED BY '"' ;
```

We are defining the structure of the table. The final lines tell Hive that incoming data will be comma separated (CSV).

If the SQL is new to you, see here: http://www.w3schools.com/sql/sql_create_table.asp

The next step, in a new Hive worksheet, is to load the data:

```
LOAD DATA INPATH '/user/maria_dev/tutorials/T100data.csv' OVERWRITE INTO TABLE T100;
```

This command moves the CSV data from the HDFS tutorials directory and puts it into the Hive table called T100, overwriting anything that might have been in there before. Also note that the CSV file is literally moved - it will no longer be in the tutorial directory.

Now prove you have some data:

```
SELECT * FROM T100 LIMIT 100;
```

So now we have two separate tables within Hive. You may notice that the airport names are different in the two tables. The thing they have in common, however, is that the Top 100 provides something it calls code, and the airports table has a column with iata_codes (http://en.wikipedia.org/wiki/International_Air_Transport_Association_airport_code). These are in fact the same thing and this will allow us to JOIN these tables based on those columns having equal values. Try this as our first multi-table query:

```
select A.name, B.passengers_2010
from airports A, T100 B
where A.iata_code = B.code ;
```

Notice that we have two tables in the FROM section, each given an alias (A for airport and B for T100). The columns required in the SELECT, and those used in the WHERE are identified by using the alias to preface the column name. This allows us to be clear about exactly which columns we are talking about and will help if we ever join tables which contain columns with the same name in them.

Now perhaps we can ask: For each country, how many airports appear in the Top 100 list? If you know SQL already, see if you can write the query that will give us the answer before looking at the code below. If you are less confident with SQL the query below will give us an answer. Try and understand the code as you use it. We used the count function earlier, and we also used the Group By clause. The WHERE clause here is being used to join the two tables together:

```
select A.iso_country, count(A.iata_code) as NumberofAirportsinTop100
from airports A, T100 B
where A.iata_code = B.code
Group by A.iso_country  ;
```

Nearly there! But the list is not very readable unless you walk around with the iata country codes in your head. Sometimes data scientists need to grab data from a variety of sources to make a finished report. In this case the easiest publicly available place to grab these codes from is Wikipedia: http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2. There are all sorts of webpage scraping tools we could use but for speed (often a driving factor!), given the small size of the table, I have just used a quick-and-dirty cut and paste technique. We talk about means for capturing and moving data in later weeks. For the time being, use the file I created, which is in Blackboard, called countrycode.csv. If you review the content using Textpad or Notepad you will see something like Figure 9.

Figure 9

We can now treat this in the same was as we did the previous CSV file and upload it to HDFS and then create a Hive table to accept the data. In my case I called this new table *countrycodes*. See if you can work through the steps to create this table for yourself. If you get stuck there are some pointers at the end of these notes, or, better still, ask one of your colleagues. Learning from peers is a very valid part of M-level study. Of course tutors will be willing to help too!

Now we need to create a three-way join so that the output has country name instead of the code. Make sure you understand this code before you use it:

```
select C.countryname, count(A.iata_code) as NumberofAirportsinTop100
from airports A, t100 B, countrycodes C
where A.iata_code = B.code
AND A.iso_country = C.countrycode
Group by C.countryname  ;
```

We will see later, when we use Flume and other tools, that we can use HiveQL against Twitter output stored in JSON format.

Hive Practice Task

As the old saying goes: *practice makes perfect*! Here is a task to test if you have understood the section on Hive. A suggested answer is at the end of the notes.

Using the data we already have, create a query that answers the question: *Which is the highest airport in the top 100*?

Help on Countrycodes

First use HDFS explorer to upload the countrycode.csv into your maria_dev\tutorials\ directory. Then use this code to create the table:

```
Create Table countrycodes
(countrycode STRING,
countryname STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ","
ESCAPED BY "" ;
```

Finally load the data:

```
LOAD DATA INPATH '/user/maria_dev/tutorials/countrycode.csv' OVERWRITE INTO TABLE
countrycodes ;
```

Suggested Solution to the Hive extra task

Using the data we already have, create a query that answers the question: *Which is the highest airport in the top 100*?
First of all find out which is highest of those airports in Top100:

```
SELECT Max(A.elevation_ft) AS maxelevation
FROM airports A, Top100 B
WHERE A.iata_code = B.code
;
```

Then we can use the join to provide the airport name. As we said in the text, a nested subquery would perhaps have been more elegant, but they are not valid in Hive as yet.

```
SELECT * FROM
(
SELECT A.name, A.elevation_ft
FROM airports A, Top100 B
WHERE A.iata_code = B.code
) t100
JOIN
(  SELECT Max(A.elevation_ft) AS maxelevation
FROM airports A, Top100 B
WHERE A.iata_code = B.code
) maxdata
ON t100.elevation_ft = maxdata.maxelevation
;
```