MIDS W205

Lab #	6	Lab Title	Apache Spark SQL - a few features
Related Module(s)	6	Goal	Get you started on spark and pyspark
Last Updated	9/24/15	Expected duration	20 to 30 minutes

This Lab (to be removed)

In this lab you will get an hands on introduction to Sparks and specifically to Spark SQL. After this lab you should be able to launch Spark, create tables, load data and run simple queries using Spark SQL. We will go over Apache Spark features and common commands as below:

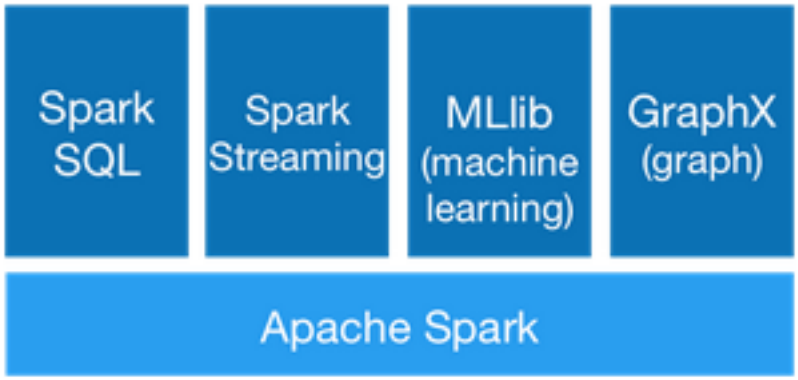
1. How to start spark
2. Basic spark operations such as creating RDDs and operations on RDDS.
3. Counting, filtering RDDs.
4. Creating key-value tuple structures on RDDS and operations on those.
5. Using Spark SQL.
6. ---
7. Accessing through Java database connectivity (JDBC)
8. Accessing through Java database connectivity - Beenline
9. Let's create a table and load data using CSV file
10. Accessing Spark-SQL in Python Code
11. Caching tables and Un-caching tables

Intro

Apache Spark

Apache Spark is an open-source distributed computing. Spark uses in-memory processing and can sometime be up to 100 times faster than regular Hadoop Map Reduce on certain types of jobs.

Apache Spark has a basic computing substrate and several frameworks built on top of it. There is a framework for querying structured data (Spark SQL), a streaming analytics framework, a machine learning framework etc.



Spark SQL is a Spark module in which uses structured data processing. It provides a programming abstraction called DataFrames. Spark SQL acts as a distributed SQL query engine.

Here are some useful resources:

Resource	What
http://spark.apache.org/docs/latest/programming-guide.html	This guide shows each of these features in each of Spark's supported languages. It is easiest to follow along with if you launch Spark's interactive shell. This includes reference to the basic commands you can perform on RDD's. Such as filter records, count records, join records.
https://spark.apache.org/docs/1.1.0/sql-programming-guide.html	Guide for using Spark SQL.
https://spark.apache.org/docs/latest/sql-programming-guide.html#running-the-spark-sql-cli	Guide tp Spark SQL CLI Shell.
https://spark.apache.org/docs/0.9.0/python-programming-guide.html	Python spark programming guide.
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.SparkContext	Programming guide for the Spark Context object. Here you can find actions available on the Spark Contexts.

Step-1. Check installation

Firstly, if an example starts with the "$" prompt, it is run in the Linux shell. If it starts with the ">>>" prompt it is run in pyspark shell.

Echo your SPARK_HOME environment variable to see where you Spark is installed.

```
%echo $SPARK_HOME
```

You should see something like:

```
$ echo $SPARK_HOME
/Applications/devtools/spark
```

Got to that directory and look what is in it.

```
$ cd $SPARK_HOME
$ ls
CHANGES.txt NOTICE      README.md   bin   data  ec2   lib   python
LICENSE     R           RELEASE     conf  derby.log   examples    old
sbin
```

If you look in bin you will see spark-shell, pyspark and other tools for running and managing spark.

In your shell profile (often .bash_profile) you may have something along the following lines so that the Spark commands are in your shell execution path.

```
export SPARK=/usr/lib/spark
export SPARK_HOME=$SPARK
export PATH=$SPARK/bin:$PATH
```

You can test to see that your shell can find them by running.

```
$ which spark-shell
$ which pyspark
$ which spark-sql
```

If your shell cannot find any of those programs you will likely need to check you installation. If which can find the commands it will return the location of the programs.

For this lab you can use any substantial data file as replacement for “Crimes_-_2001_to_present.csv” but you need to modify the commands accordingly. You can down load the “Crimes_-_2001_to_present.csv” file from github. Because it is a big file and we uploaded it to github we needed to compress and spit the file. So you will need merge the parts and uncompress the result.

The instructors may have the files more easily accessible for you if you have problems with the below.

Get the files in this directory

https://github.com/UC-Berkeley-I-School/w205-labs-exercises/tree/master/data

Once you have the files (should be seven of them all starting with the letter “x”) run the following commands (make sure you do not have other files starting with x in the directory).

```
$ cat x* > Crimes_-_2001_to_present.csv.gz
$ gunzip Crimes_-_2001_to_present.csv.gz
```

the result should be that you have the Crime data in a csv file in your directory. If you run ls it should look something like this:

```
$ ls
Crimes_-_2001_to_present.csv  xac                 xaf
xaa                       xad                     xag
xab                       xae
```

What you should have learnt

You should have checked the basic installation and that the Spark programs can be found by your interactive Linux (Unix) shell.

Step-1. Start pyspark

First we will start a spark shell so that we can access spark and interactively process sprak commands. We will be using pyspark, which is a python based shell for spark.

Start the shell assuming you have the spark bin directory in your PATH environment variable.

```
pyspark
```

Otherwise go to the /bin directory in the installation folder and type.

```
./pyspark
```

In the pyspark shell you can use python instructions. Create a variable with some value using this command.

```
>>> x = [1,2,3,4,5,6,7,8,9];
>>> print x
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> len(x)
```

```
9
```

There is something call a spark context. The spark context is the “object” you use to refer to the spark cluster. type sc to verify you have one:

```
>>> sc
<pyspark.context.SparkContext object at 0x1063b3410>
```

So far you only used Python statements. You can use the Spark context to create a Spark RDD from this data using the following command:

```
>>> distData = sc.parallelize(x);
>>> print distData;
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:391
```

The Spark Context parallelize action is used to take local programming collections and create RDDs from them. In this case we created a RDD from a Python array.

distData is an RDD representation. Try doing len(distData), what happens and why? To count elements in an RDD you need to use RDD actions. You can find a list of actions in the programming guide. To count the elements you use the count() action. Try the following:

```
>>> nx=distData.count()
>>> print nx
9
```

The default level of logging can be distracting. To reduce the logging information go to the `$SPARK_HOME/conf` directory. Create a `log4j.properties` file. If you do not already have one you can do that by copying the `log4j.properties.template` file. Change the logging level to warnings only by change `INFO` to `WARN` in the property `log4j.rootCategory=WARN, console`

```
# Set everything to be logged to the console
log4j.rootCategory=WARN, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
```

If you rerun the commands above you should see much less output in the Spark shell.

What you should have learnt

You should understand how to start pyspark and that pyspark is a shell with access to the Python language and to the underlying Spark cluster capabilities. We also show you how to make pyspark less verbose by reducing the amount of logging displayed. If you have problems you may consider increasing logging again to understand what is going on in your execution.

Step-2. Load a file and count the rows.

Spark is commonly used to process large sets of data, and naturally we often read these data files from disc. The action textFile is an operation on Spark Contexts that creates a RDD from a file that resides in HDFS or the local files system. Make sure you cloned the data file Crimes_-_2001_to_present.csv in the lab directory. Now create an RDD from that file using this action:

```
crimedata =sc.textFile("Crimes_-_2001_to_present.csv")
```

Print the number of lines this RDD using the following command:

```
>>> print crimedata.count()
5862796
>>>
```

As you can see there are almost 6 million records, so it took a few seconds to count them.

You can get the first element of the RDD with the operation first:

```
>>> crimedata.first()
```

You can get the n first elements with the operation take(n)

```
>>> crimedata.take(10)
```

As you see the data include the header information. Remove the header from the RDD is not straight forward. The reason is that RDD is immutable by design. One way to remove it is to create a new RDD and to filter the first row out. There are many ways to do that, here is an example:

```
>>> noHeaderCrimedata = crimedata.zipWithIndex().filter(lambda
(row,index): index > 0).keys()
```

It is a quite a processing heavy way of doing it, but as we mentioned RDD's are immutable you can not just go in and remove a record. It is an inherent assumption to Spark which allows it to certain things more efficiently.

Print the first line to check that the header if gone. And count the lines to make sure the number seems correct.

```
>>> noHeaderCrimedata.first()
>>> noHeaderCrimedata.count()
```

Here is a more Python like way of doing the same, that may be easier to understand. We first define a Python function in our Python spark shell using the following line.

```
>>> def remove_header(itr_index, itr): return iter(list(itr)[1:]) if
itr_index == 0 else itr
```

We then execute a mapPartitionWithIndex operations passing that function as an argument.

```
>>> noHeaderCrimeData2 = crimedata.mapPartitionsWithIndex(remove_header)
```

Print the first line and the count to make sure it is correct.

What you should have learnt

You should understand how to create an RDD from a file, how apply operations on RDD. You used examples such as first and count. We also illustrated that RDD's are immutable and that to even remove one row (the header) you need to create a new RDD.

Step-3. Filter records, structures

One obvious operation for Spark is to filter the data. Lets filter out all crimes that seems to be related to "NARCOTICS".

We can do this using the filter operations and a lambda function that checks if the word "NARCOTICS" appears in the each row. We will only return rows that included that word.

```
narcoticsCrimes = noHeaderCrimedata.filter(lambda x: "NARCOTICS" in x)
>>> narcoticsCrimes.count()
663712
```

It appears that the are 663712 crimes related to narcotics. Use take(n) to check that the data seems ok. For example:

```
>>> narcoticsCrimes.take(20)
```

The RDD we have are just long strings, the fact that fields are comma separated does not mean anything to the Spark RDD. If we want to create a structure with which we want to do some more advanced things with we need to parse the rows and create the appropriate structure. The operations below splits each record up as an array using the Python operation split. It then create a new RDD were each row is an array of strings as opposed to one long string.

```
>>> narcoticsCrimeRecords = narcoticsCrimes.map(lambda r :
r.split(","))
```

You can see the first array record using:

```
>>> narcoticsCrimeRecords.first()
```

You can check that you still have the same number of rows using:

```
>>> narcoticsCrimeRecords.count()
```

What you should have learnt

You should understand that RDD's are immutable. You can filter RDD's but it creates a new RDD. You should also understand that RDD's do not understand anything about the structure of the records (except for key-value structures which we discuss in the next section). But you can store any Python structure that seems useful in an RDD.

Step-3. Key-values

An important structure in spark is called Key Value pairs. In Python those are represented as Python tuples. A tuples is an immutable sequence of elements of various types.

You can create a new RDD consisting of tuples using the following operation:

```
>>> narcoticsCrimeTuples = narcoticsCrimes.map(lambda x:
(x.split(",")[0], x))
```

You can check that the number of tuples is the same as the number of records in the data.

```
>>> narcoticsCrimeTuples.count()
```

It takes the first first element and makes it a key and the rest if the row becomes the value part of the tuple. You can examine the tuple using the following operations:

```
>>> narcoticsCrimeTuples.first()
```

And you can check it out using these RDD and Python functions.
Get the first tuple:

```
>>> firstTuple=narcoticsCrimeTuples.first()
```

How many elements do you have in the tuple?

```
>>> len(firstTuple)
```

What is the key of the first tuple?

```
>>> firstTuple[0]
```

What is the value of the first tuple?

```
>>> firstTuple[1]
```

There are many operations you can do once you have a key-value tuple. You can join, reduce, map etc. You can read about the operations in the RDD Spark programming guide. One operation you can do is to sort by key:
>>> sorted=narcoticsCrimeTuples.sortByKey()

If print the first element in the sorted RDD and the original RDD you will see they are different.

```
>>> sorted.first()

>>> narcoticsCrimeTuples.first()
```

What you should have learnt

Now you should understand the concept of Key-Value tuples, and understand how you can create them. You have also tried one operation on RDD's using the key-value structure.

Step-4. Start Spark-SQL

Spark SQL can be used directly from pyspark or a scala shell. But there is also an Spark SQL CLI called the Beeline client. If you use pyspark or use Spark SQL programmatically you need create a special Spark SQL contexts. With the Spark SQL CLI the context is already there for you and you can use SQL commands.

You start Beeline with the command

```
$spark-sql
```

Once started you can run some commands to see that it works. Show tables, create a table and drop the table by running the commands below.

```
spark-sql> show tables;

spark-sql> create table dummy (somedata  varchar(500));
OK
Time taken: 0.369 seconds

spark-sql>  show tables;
dummy false
Time taken: 0.053 seconds, Fetched 1 row(s)

spark-sql>  drop table dummy;

spark-sql>  show tables;
```

You should see the following result

You should understand the difference between using the Spark SQL CLI and using Spark SQL programmatically. You are able to start Spark SQL CLI and issues some basic commands to see that it works.

Step-5. Let's create a table and load data using CSV file:

You can create a Spark SQL table. The following create statement creates a table that has a schema that corresponds to the web_session_log data. Run the following create statement directly on the spark-sql shell prompt.

```
create table Web_Session_Log
(DATETIME varchar(500),
USERID varchar(500),
SESSIONID varchar(500),
PRODUCTID varchar(500),
REFERERURL varchar(500))
row format delimited fields terminated by '\t'
stored as textfile;
```

You can load files from the local files system or from HDFS. Lets load a the web_log data available on github.

Assuming you have the weblog data in the directory were you are running spark-sql shell, you can load the file from the files system into the table using the command below. If the file is located somewhere else you need to modify the path ot the file.

```
spark-sql>  LOAD DATA LOCAL INPATH "./weblog_lab.csv" INTO TABLE
web_session_log;
```

Once the data is loaded you can count the number of rows. You can count the number of rows with the following select statement.

```
spark-sql> select count(*) from web_session_log;
```

You can check that this seems reasonable by comparing with the number of rows in the original file. You can get the number of files using the Unix/Linux command wc.

```
$ wc -l weblog_lab.csv
```

Using Spark SQL you can use a number of SQL command so query your data. Lets select all rows in the web log that are related eBay.

```
spark-sql> select * from web_session_log where refererurl =
"http://www.ebay.com" ;
```

And now lets count the number of rows that are related to eBay.

```
spark-sql> select count(*) from web_session_log where refererurl =
"http://www.ebay.com" ;
```

You should get that there are 3943 entries related to eBay in this data set.

You should see the following result

In this section you should have learnt how to create a table in the Spark SQL CLI and how to load data into the empty table. You also practiced some simple SQL commands on the loaded data set.

---- Below needs to be refined/reviewed:

Step-6. Accessing Spark-SQL in Python Code:

Programming Spark SQL with Python

Here is a sample code:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
Running SQL Queries Programmatically :
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df = sqlContext.sql("SELECT * FROM Web_Session_Log limit 10")
Programmatically Specifying the Schema :
from pyspark.sql import *
sqlContext = SQLContext(sc)
lines = sc.textFile("/mnt/weblog.csv")
parts = lines.map(lambda l: l.split("\t"))
Web_Session_Log = parts.map(lambda p: (p[0], p[1],p[2], p[3],p[4]))
schemaString = "DATETIME USERID SESSIONID PRODUCTID REFERERURL"
fields = [StructField(field_name, StringType(), True) for field_name in
schemaString.split()]
schema = StructType(fields)
schemaPeople = sqlContext.createDataFrame(Web_Session_Log, schema)
schemaPeople.registerTempTable("Web_Session_Log")
results = sqlContext.sql("SELECT USERID FROM Web_Session_Log")
names = results.map(lambda p: "Name: " + p.USERID)
for name in names.collect():
print name
```

How to Run This Python Code:

First, go to pyspark prompt by running ./pyspark from your installation directory.

```
[root@ip-10-229-95-146 bin]# ./pyspark
```

You will see the following prompt:

```
Using Python version 2.6.6 (r266:84292, Jan 22 2014 09:42:36)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

You can just launch pyspark from Spark installation bin directory and paste above code. Or, you can create a file with the above content as say mypyspark.py and run as

```
./bin/pyspark /yourdirectory/myspark.py
```

Analyze the result. Notice that the table used in this step was created in step 4. If you have created your own table, refer to that. If you run into any issue, you will need to debug. This will help you to learn what each line of code is doing.

You should see the following result

Step-7. Caching tables and Un-caching tables;

Caching tables

This is extremely useful when you are joining tiny dataset with huge dataset.

CACHE TABLE and UNCACHE TABLE statements are available to do the above making it very easy.

```
To Cache a table :
CACHE TABLE logs_last_month;
To UnCache a table :
UNCACHE TABLE logs_last_month;
```

Once as table is cached, you can use in your spark queries.

You should see the following result

Step-X. Accessing through Java database connectivity (JDBC)

Starting a JDBC server

Spark SQL provides JDBC connectivity, which is useful for connecting business intelligence (BI) tools to a spark cluster and other users or applications.

Here are the steps to launch a the Spark SQL JDBC server

From Spark Installation folder

```
cd /usr/lib/spark

./sbin/start-thriftserver.sh –master sparkMaster
```

You should see the following result

Step-X. Let's connect to the above server(JDBC) from Step-2 through Beenline:

Connecting to a Database

From Spark Installation folder

```
cd /usr/lib/spark

//10000 will be the default thrift server port

./bin/beenline -u jdbc:hive2://localhost:10000

0: jdbc:hive2://localhost:10000> show tables;
```

You should see the following result

What you should have learnt

1. What are the various components of Apache Spark?
2. What is the difference between Spark-shell and Spark-sql?
3. What is a RDD?
4. Why Spark is faster than Map-reduce FW?
5. How will you use Spark from your visualization tools?

Troubleshooting

If you get an exception looking like “ERROR SparkContext: Error initializing SparkContext. java.net.UnknownHostException:…” make sure you have the make of our computer added to the /etc/hosts file. For example add the line “127.0.0.1 <myhost>”, where <myhost> is the name of our computer.