New Data Lake/Tutorial 4 ...

Tutorial 4: Working with the Spark Interpeter

This tutorial was built for BDCS-CE version 17.4.1 as part of the New Data Lake User Journey: here (https://github.com/oracle/learning-library/tree/master/workshops/journey2-new-data-lake). Questions and feedback about the tutorial: david.bayard@oracle.com (mailto:david.bayard@oracle.com)

```
Be sure you previously ran the Tutorial: "Citi Bike New York Demo Introduction and Setup"
```

This tutorial will illustrate how to run Spark interpreter to define a temporary Spark SQL table against a CSV file and then query it.

Contents

As a reminder, the documentation for BDCS-CE can be found: here (http://docs.oracle.com/cloud/latest/big-data-compute-cloud/index.html)

About Spark and Spark SQL

BDCS-CE version 17.4.1 comes with Spark version 2.1 and Scala version 2.11. This BDCS-CE version supplies Zeppelin interpreters for Spark(Scala), Spark SQL, SparkR, and pySpark. This tutorial will give you examples using the Spark(scala) and SparkSQL interpreters.

The tutorial assumes you have a basic knowledge about Spark. To learn more about Spark, check out Spark Quick Start (https://spark.apache.org/docs/2.1.0/quick-start.html) and Spark SQL Programming Guide (https://spark.apache.org/docs/2.1.0/sql-programming-guide.html)

Reading the data and registering as a Spark SQL table

The next step is to use Spark to read our bike data CSV file that we uploaded to the Object Store. Once we read the CSV into a Spark Data Frame, we will ask Spark to cache the data in memory. Then we will register the data frame as a Spark SQL temp table.

You can review the Spark SQL programming guide for a refresher about Data Frames and Temporary Tables: Spark SQL Programming Guide (https://spark.apache.org/docs/2.1.0/sql-programming-guide.html)

Spark Scala to read CSV and register as a temp view (takes 1-3 minutes)

```
%spark

//a previous tutorial placed the csv file into your Object Store citibike container
//notice the use of the swift://CONTAINER.default/ syntax
val Container = "journeyC"
val Directory = "citibike"

sqlContext.setConf("spark.sql.shuffle.partitions", "4")

//val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("swift://"+Container+".default/"+Directory+"/
//We will use the bdfs (alluxio) cached file system to access our object store data...
val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema","true").load("bdfs://localhost:

// If you get this error message:
// java.lang.IllegalStateException: Cannot call methods on a stopped SparkContext.
// Then go to the Settings tab, then click on Notebook.  Then restart the Notebook.  This will restart your SparkContext


println("Here is the schema detected from the CSV")
df.printSchema()
println("..")

println("# of rows: %s".format(
  df.count()
))
println("..")
```

```
Container: String = journeyC
Directory: String = citibike
df: org.apache.spark.sql.DataFrame = [Trip Duration: int, Start Time: timestamp ... 13 more fields]
Here is the schema detected from the CSV
root
 |-- Trip Duration: integer (nullable = true)
 |-- Start Time: timestamp (nullable = true)
 |-- Stop Time: timestamp (nullable = true)
 |-- Start Station ID: integer (nullable = true)
 |-- Start Station Name: string (nullable = true)
 |-- Start Station Latitude: double (nullable = true)
 |-- Start Station Longitude: double (nullable = true)
 |-- End Station ID: integer (nullable = true)
 |-- End Station Name: string (nullable = true)
 |-- End Station Latitude: double (nullable = true)
 |-- End Station Longitude: double (nullable = true)
 |-- Bike ID: integer (nullable = true)
 |-- User Type: string (nullable = true)
 |-- Birth Year: integer (nullable = true)
 |-- Gender: integer (nullable = true)
..
```

Querying the bike trip information

Now we can show some examples of querying our bike_trips table. You will need to have first run the above paragraph to ensure that the temporary table bike_trips_temp is registered in your current Spark Session.

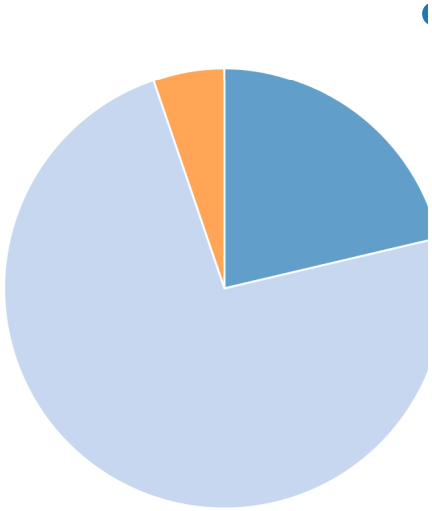
Trips by Gender

```
%sql
select
 case when a.gender=1 then 'Male' when a.gender=2 then 'Female' el
        a.trip_count
from (select gender, count(*) trip_count from bike_trips_temp
group by gender) a
```

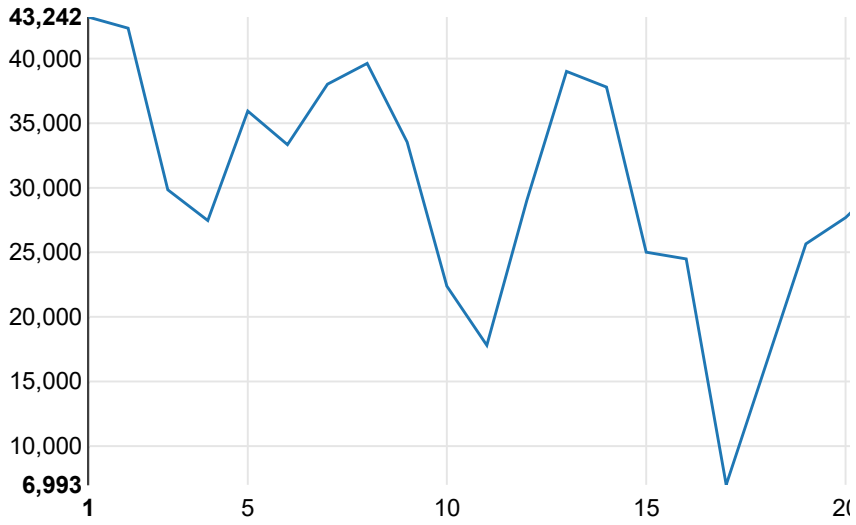
Trips by Day of Month

```
%sql
select dayofmonth, count(*)
from (select date_format(`Start Time`,"H") hour,
 date_format(`Start Time`,"E") dayofweek,
 date_format(`Start Time`,"d") dayofmonth
from bike_trips_temp) bike_times
group by dayofmonth
```

settings



settings



Trips by Day of Week and Gender

```
%sql
select dayofweek, count(*)
from (select date_format(`Start Time`,"H") hour,
 date_format(`Start Time`,"E") dayofweek,
 date_format(`Start Time`,"d") dayofmonth,
 case when gender=1 then 'Male' when gender=2 then 'Female' else '
from bike_trips_temp) bike_times
where (gender="${gender=Male,Male|Female|unknown}" )
group by dayofweek
```

gender

Male

settings

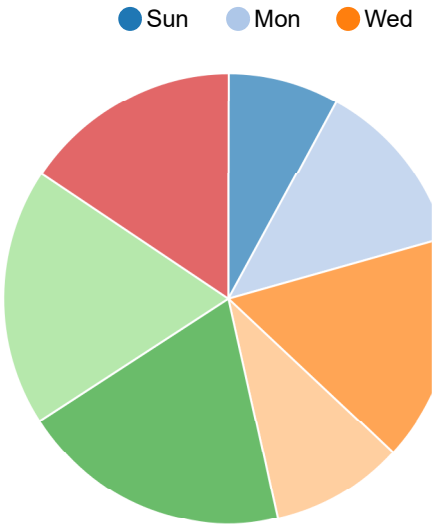
Bike Trips by Hour by day of week

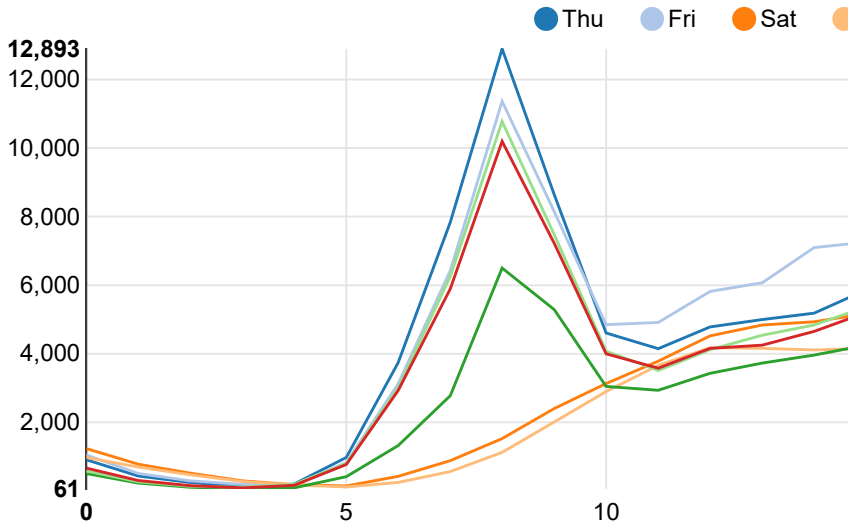
```
%sql
select dayofweek, hour, count(*)
from (select date_format(`Start Time`,"H") hour,
 date_format(`Start Time`,"E") dayofweek,
 date_format(`Start Time`,"d") dayofmonth,
 case when gender=1 then 'Male' when gender=2 then 'Female' else '
from bike_trips_temp) bike_times
where (gender="${gender=Male,Male|Female|unknown}" )
group by dayofweek, hour
```

gender

Male

settings





Next Steps

So far, we have downloaded Citi Bike data and stored it into the Object Store. Then, we configured Spark to be able to work with CSV files. Then, we read in the data and defined a Spark SQL temporary table with it. Finally, we demonstrated a number of different queries. Did you notice any patterns? For instance, that men use Citi Bikes more than women? That on workdays (Mon-Fri) there is a peak around 8am and 5pm, but that peak does not exist on Saturday and Sunday?

Here are some suggested next steps:

- Run the Demonstration Presidental Speeches with Spark and Spark SQL
- Explore the Tutorial Spark and Maps in Zeppelin
- Upload some of your own data to the Object Store and experiment with Spark and Spark SQL

Extra Credit - Saving our temporary Spark SQL table as a permanent Hive table

The next few paragraphs show you how you can save a copy of the Spark SQL temporary table as a new permament Hive table. This might be useful if you want to use BI tools, like Oracle Data Visualization Desktop, to query the permanent table.

The only trick is that Hive doesn't like spaces in column names, so we rename our columns in our Create Table as Select statement below.

Query to list our hive tables BEFORE

```
%sql
show tables
```

database	tableName
default	bike_trips
default	bike_trips_objectstore
default	bike_trips_small
	bike_trips_temp

HiveQL to drop the table (in case you already ran the next step)

```
%sql
drop table bike_trips_parquet
```

HiveQL to create a permanent Hive table from our SparkSQL temporary view (this will have no output)

```
%sql
create table bike_trips_parquet
stored as parquet
as select `Trip Duration` TRIPDURATION,
`Start Time` STARTTIME,
`Stop Time` STOPTIME,
`Start Station ID` STARTSTATIONID,
`Start Station Name` STARTSTATIONNAME,
`Start Station Latitude` STARTSTATIONLATITUDE,
```

Query to show our new permanent table in action

```
%sql
select * from bike_trips_parquet limit 5
```

TRIPDURATION	STARTTIME	STOPTIME	STARTSTATIONID	STARTSTATIONNAME	STARTSTATIONLA

TRIPDURATION	STARTTIME	STOPTIME	STARTSTATIONID	STARTSTATIONNAME	STARTSTATIONLA
528	2016-12-01 00:00:04.0	2016-12-01 00:08:52.0	499	Broadway & W 60 St	40.76915505
218	2016-12-01 00:00:28.0	2016-12-01 00:04:06.0	3418	Plaza St West & Flatbush Ave	40.6750207
399	2016-12-01 00:00:39.0	2016-12-01 00:07:19.0	297	E 15 St & 3 Ave	40.734232
254	2016-12-01 00:00:44.0	2016-12-01 00:04:59.0	405	Washington St & Gansevoort St	40.739323
1805	2016-12-01 00:00:54.0	2016-12-01 00:31:00.0	279	Peck Slip & Front St	40.707873

Query to list our Hive tables AFTER

```
%sql
show tables
```

database	tableName
default	bike_trips
default	bike_trips_objectstore
default	bike_trips_parquet
default	bike_trips_small
	bike_trips_temp

Change Log

November 15, 2017 - replaced registerTempTable with createOrReplaceTempView
October 11, 2017 - Added spark.sql.shuffle.partitions=4
September 7, 2017 - Confirmed it works with 17.3.5-20. Switched filepath to bdfs
August 23, 2017 - Minor tweaks
August 13, 2017 - Confirmed it works with 17.3.3-20.
August 11, 2017 - Journey v2. Confirmed it worked with Spark2.1
July 28, 2017 - Confirmed it works with 17.3.1-20.

```
%md
```