AN INDUSTRIAL TRAINING REPORT

on

Data Analytics using Cloudera Hadoop

Submitted by

Harshit Khare
Roll No: 161500229

Department of Computer Engineering & Applications
Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA
August, 2018

Department of Computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

Declaration

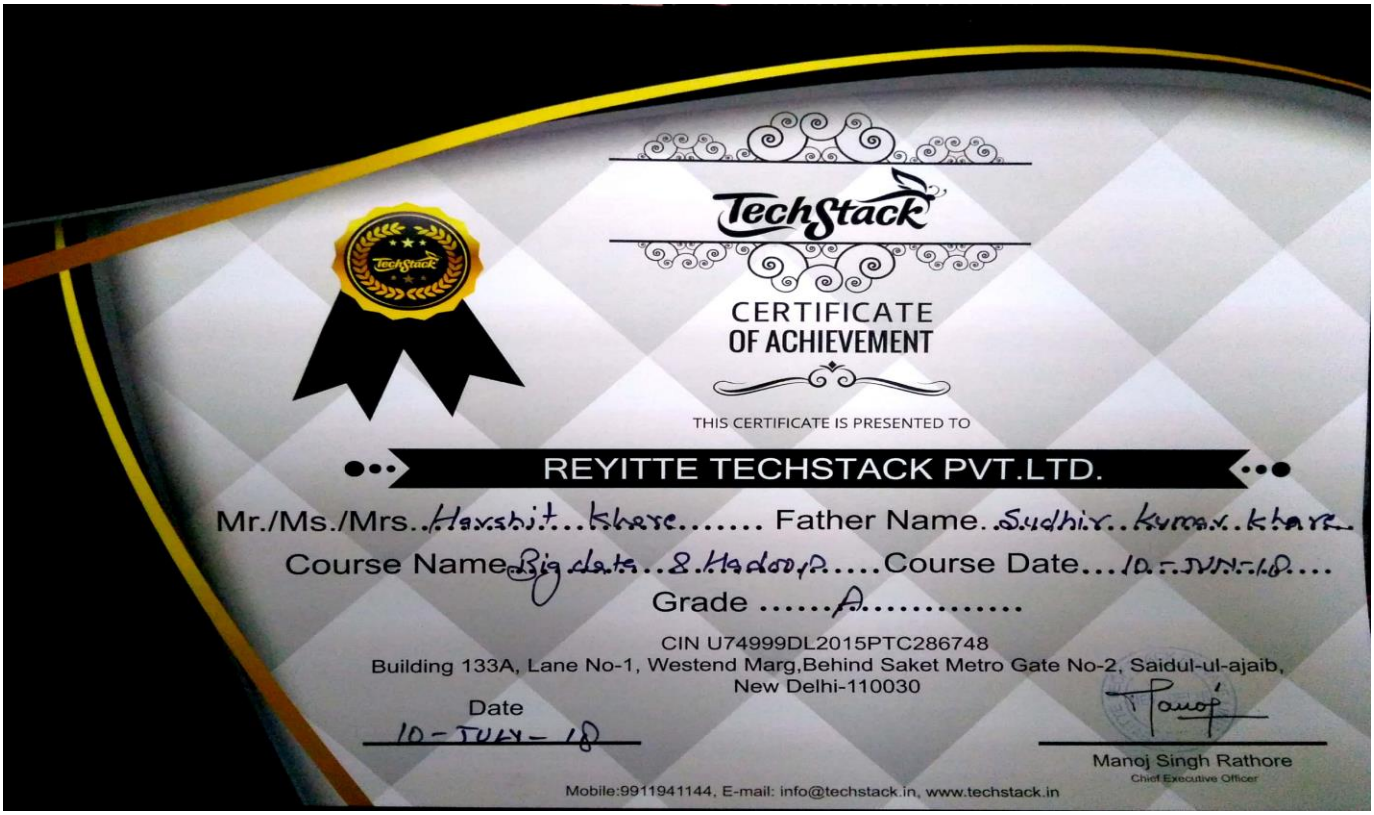
I hereby declare that the work which is being presented in the Training Project Report titled **"Data Analytics using Cloudera Hadoop",** in partial fulfillment of the requirements for Industrial Project is an authentic record of my own work carried under the supervision of **Mr. Prashant Gangwar, TechStack, New Delhi**.

Sign _____

Name of Candidate: Harshit Khare

University Roll No.: 161500229

Certificate



Acknowledgement

I would like to express my sincere gratitude to my supervisor Mr. Prashant Gangwar for providing his invaluable guidance, comments and suggestions throughout the course of the project. I would specially thank Mr. Saurabh Anand sir for constantly motivating me to work harder.

Also, I would like to thank my fellow colleagues for their assistance in the Cloudera Hadoop Framework tools & their working.

Lastly, I would like to thank my parents for constantly supporting me in the times when everything used to go wrong.

Abstract

The project consists of 4 sub-projects which uses different technologies:

Part 1: Map-Reduce: Finding words present in different files

In this part, we find which words are present in how many files along with the file names using Map-Reduce Framework on top of HDFS.

Part 2: Sqoop Task: Loading Data from RDBMS to HDFS

In this part, we load data from RDBMS (MySQL) to HDFS & then load data from HDFS to Hive tables using Apache Sqoop.

Part 3: Stocks Analysis using Hive

In this part, we run Hive queries on the stocks dataset loaded using Apache Sqoop to understand it better and then perform analysis on it.

Part 4: Twitter's Top 10 popular Hashtag Streaming per second using Apache Spark

In this part, we find Top 10 popular Hashtag on Twitter and perform Web Scraping using Spark & Scala to stream the data on per second basis.

Table of Contents

Declaration & Certificate	ii
Acknowledgement	iii
Abstract	iv
1. Introduction	**1**
1.1 Overview and Motivation	1
1.1 Introduction to Big Data	1
1.2 Introduction to Hadoop	3
1.3 Map-Reduce	5
1.4 Introduction to Sqoop	5
1.5 Apache Hive	6
1.6 Apache Spark	7
2. Company Profile	**9**
2.1 About Company	9
2.2 Company's Mission	9
2.3 Company's Philosophy	9
3. Implementation and User Interface	**10**
3.1 Map-Reduce: Finding words present in different files	10
3.2 Sqoop Task: Loading Data from RDBMS to HDFS	14
3.3 Stocks Analysis using Hive	15
3.4 Twitter's Top 10 popular Hashtag Streaming per second using Apache Spark	26
References	**30**

1. Introduction

1.1 Overview and Motivation

Working on Big Data is quite challenging due to its various implications like:

- Very large volume of Data
- Data will be spread across multiple machines
- Data will be in different formats: Structured, CSV, RDBMS, Log files, Unstructured, Data extracted from web pages & Email content

Also moving data to databases is very expensive. Daily terra bytes of data are to be uploaded which is cumbersome and more error prone.

So, a possible solution could be to analyze the data in the format they are i.e. a text file need not be uploaded into database to analyze it. Thus, data need not be uploaded into any system.

The trick is not to move the data out of the box. Instead move the code to the box where data resides. The size of the code is very less when compared to the data. Thus network contention problem is solved.

The Map-reduce framework implements the solution that we saw in the previous slide

HDFS is very similar to a file system, except that files are replicated to multiple machines for availability and scalability.

1.2 Introduction to Big Data

Big data is data sets that are so big and complex that traditional data-processing application software are inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. There are a number of concepts associated with big data: originally there were 3 concepts volume, variety, velocity. Other concepts later attributed with big data are veracity (i.e., how much noise is in the data) and value.

Characteristics of Big Data

Big data can be described by the following characteristics:

- **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.
- **Variety:** The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus, it completes missing pieces through data fusion.
- **Velocity:** In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.
- **Veracity:** The data quality of captured data can vary greatly, affecting the accurate analysis.

Why is Big Data Important?

The importance of big data doesn't revolve around how much data you have, but what you do with it. You can take data from any source and analyze it to find answers that enable

1) cost reductions
2) time reductions
3) new product development and optimized offerings
4) smart decision making.

Big Data Challenges

While big data holds a lot of promise, it is not without its challenges.

- First, big data is…big. Although new technologies have been developed for data storage, data volumes are doubling in size about every two years. Organizations still struggle to keep pace with their data and find ways to effectively store it.

- But it's not enough to just store the data. Data must be used to be valuable and that depends on curation. Clean data, or data that's relevant to the client and organized in a way that enables meaningful analysis, requires a lot of work. Data scientists spend 50 to 80 percent of their time curating and preparing data before it can actually be used.

- Finally, big data technology is changing at a rapid pace. A few years ago, Apache Hadoop was the popular technology used to handle big data. Then Apache Spark was introduced in 2014. Today, a combination of the two frameworks appears to be the best approach. Keeping up with big data technology is an ongoing challenge

1.3 Introduction to Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

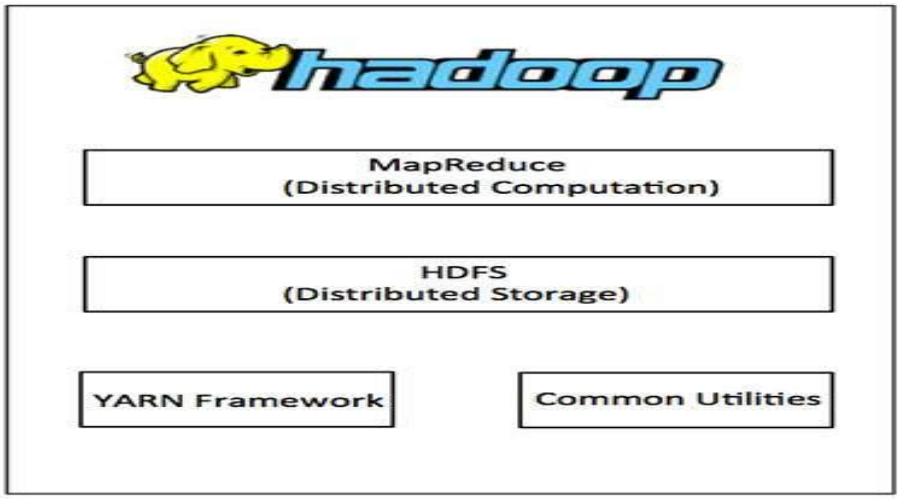
Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.

- **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.

- **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.



Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode. HDFS provides a shell like any other file system and a list of commands are available to interact with the file system.

1.4 Map-Reduce

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change.

1.5 Introduction to Sqoop

Apache Sqoop is a tool in Hadoop ecosystem which is designed to transfer data between HDFS (Hadoop storage) and relational database servers like MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc. Apache Sqoop imports data from relational databases to HDFS, and exports data from HDFS to relational databases. It efficiently transfers bulk data between Hadoop and external datastores such as enterprise data warehouses, relational databases, etc.

This is how Sqoop got its name – “SQL to Hadoop & Hadoop to SQL”.

Additionally, Sqoop is used to import data from external datastores into Hadoop ecosystem’s tools like Hive & HBase.

Why Sqoop?

For Hadoop developer, the actual game starts after the data is being loaded in HDFS. They play around this data in order to gain various insights hidden in the data stored in HDFS.

So, for this analysis the data residing in the relational database management systems need to be transferred to HDFS. The task of writing MapReduce code for importing and exporting data from relational database to HDFS is uninteresting & tedious. This is where Apache Sqoop comes to rescue and removes their pain. It automates the process of importing & exporting the data.

Sqoop makes the life of developers easy by providing CLI for importing and exporting data. They just have to provide basic information like database authentication, source, destination, operations etc. It takes care of remaining part.

Sqoop internally converts the command into MapReduce tasks, which are then executed over HDFS. It uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism.

1.6 Apache Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data and makes querying and analyzing easy.

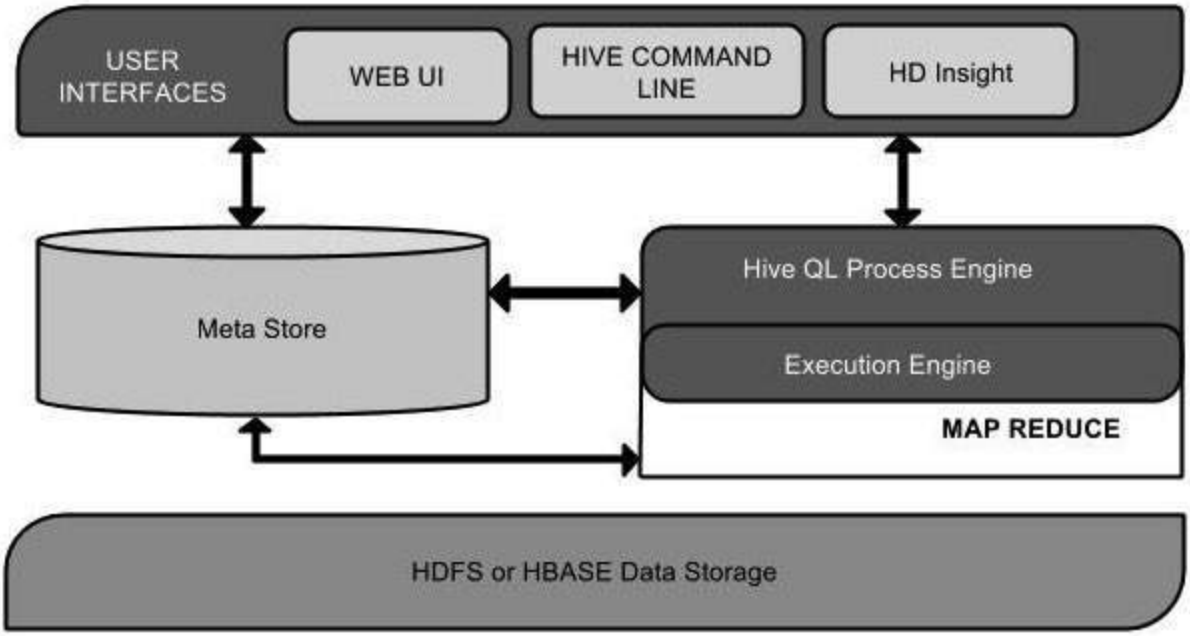
Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not:

- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Architecture of Hive

The following component diagram depicts the architecture of Hive:



1.7 Apache Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Features of Apache Spark

Apache Spark has following features.

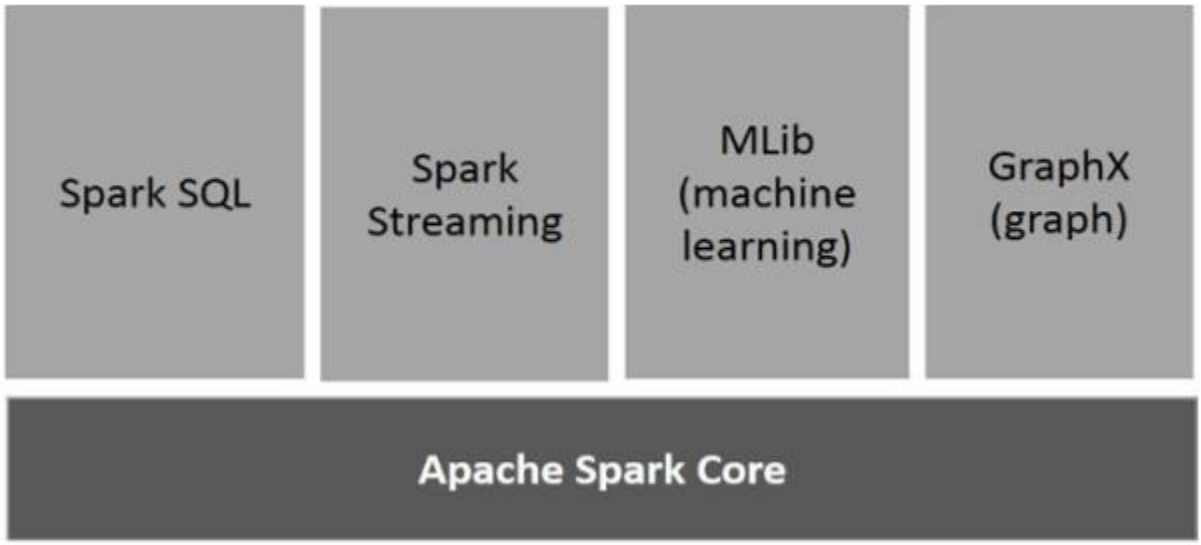
- **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

- **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

- **Advanced Analytics** − Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Components of Spark

The following illustration depicts the different components of Spark.



Resilient Distributed Datasets (RDD)

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations.

2. Company Profile

2.1 About Company

TechStack is a training institute which provides digital marketing course, big data Hadoop course, web designing course and web development course. It started in 2015 by Mr. Manoj Singh Rathore who has an aim to aware most of youth about Digital marketing in all over India.

2.2 Company's Mission

We are committed to deliver best education & career opportunities for the students as well as for working professionals. TechStack is a complete training Institute in Delhi. TechStack Pvt. Ltd in Delhi offers training courses in Digital marketing, Big Data Hadoop, Web development, Web Designing, Graphics Designing, Asp.net, Java, PHP, WordPress, with strong base of placement support, real-time trainers with working experience in MNC companies, tailor-made training curriculum to meet the career objective of the students and working professionals.

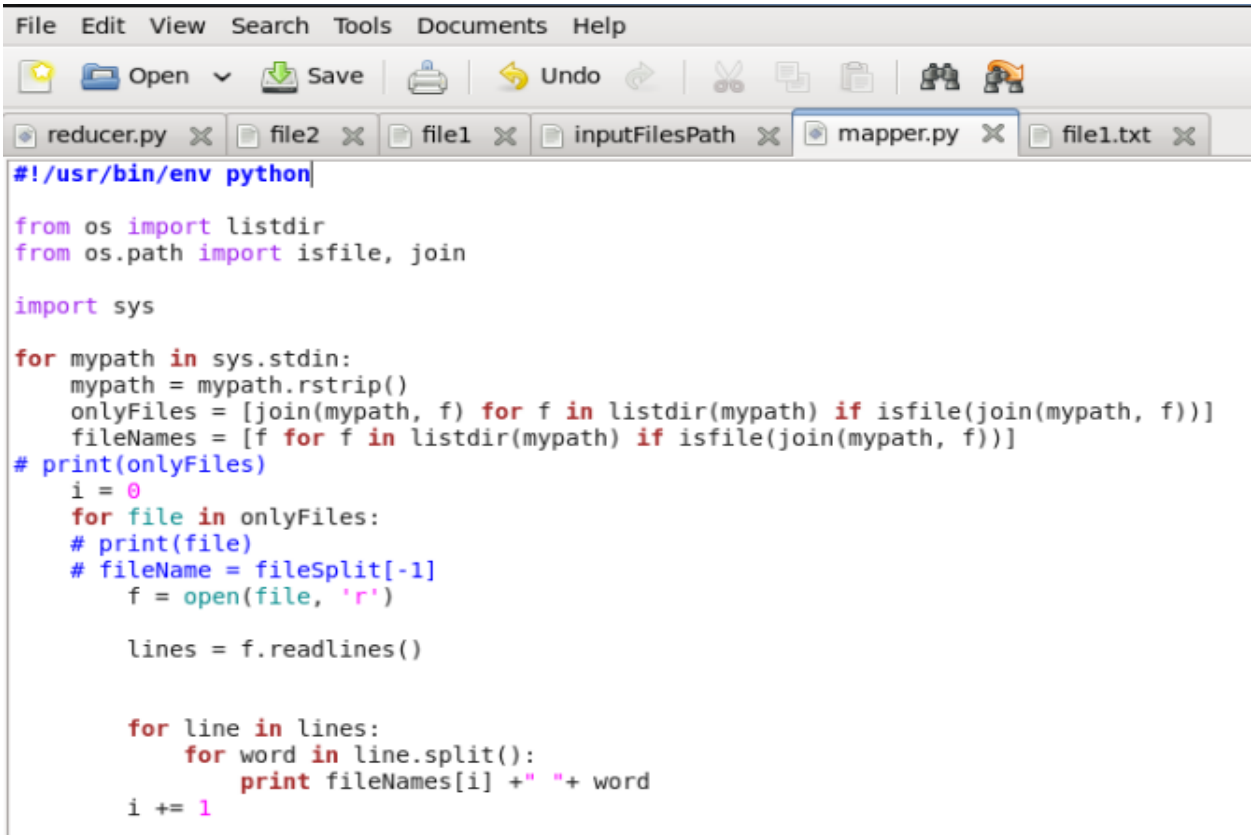
2.3 Company's Philosophy

TechStack Training in Delhi has strong network of experienced real time MNC Reyitte TechStack- is a professional training Company offering IT & Non-IT enabled Advance trainings for B.E., B-Tech, MCA, BCA, MSc and MBA fresher's and experienced Developers/programmers in various platforms. Deserving candidates may be awarded scholarships and other benefits, depending on their caliber.

3. Implementation and User Interface

3.1 Map-Reduce: Finding words present in different files

mapper.py



reducer.py

reducer.py | file2 | file1 | inputFilesPath | ma

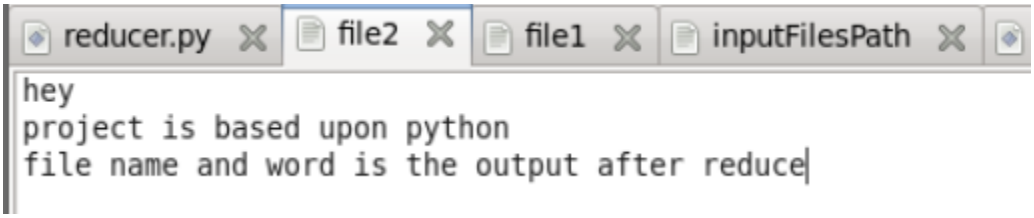
```
#!/usr/bin/env python

import sys

def removeDuplicates(words):
    removed = []
    for word in words:
        if word not in removed:
            removed.append(word)

    return removed
wordKeyFileValue = {}
final = []
fileAndWord = {}
words = []
for line in sys.stdin:

    line = line.strip()

    file, word = line.split()
    words.append(word)

    if file not in fileAndWord:
        fileAndWord[file] = []
        fileAndWord[file].append(word)
    else:
        if word not in fileAndWord[file]:
            fileAndWord[file].append(word)

    removed = removeDuplicates(words)

    # print(removed)

    # print(fileAndWord)

    #
    for word in removed:
        for key in fileAndWord.keys():
            if word in fileAndWord[key]:
                final.append(key)
        wordKeyFileValue[word] = final
        final = []
    # print(wordKeyFileValue)
    words = sorted(wordKeyFileValue.keys())
    values = wordKeyFileValue.values()
for word, values in zip(words, values):
    print ""
    print word,
    for value in values:
        print value,
    print ""
    # print(fileAndWord)
```

file1.txt

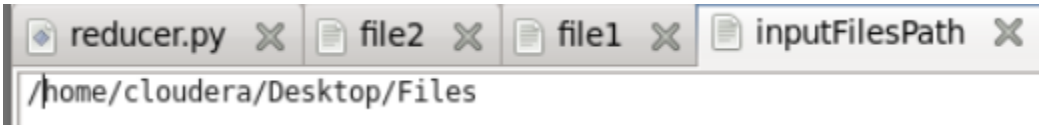
file2 | file1 | inputFilesPath | mapper.py | file1.txt

```
: is based on map reduce
has two methods - map and reduce
```

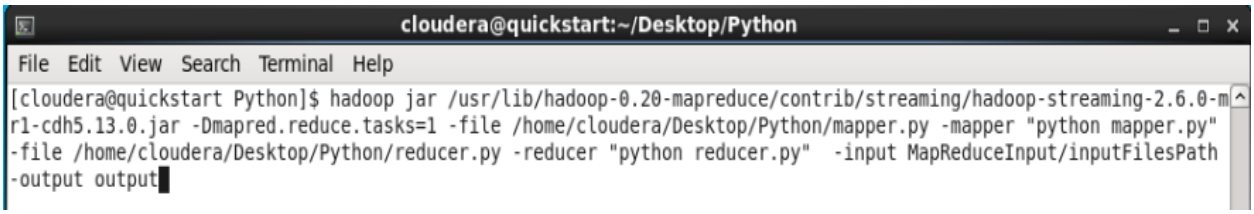
file2.txt



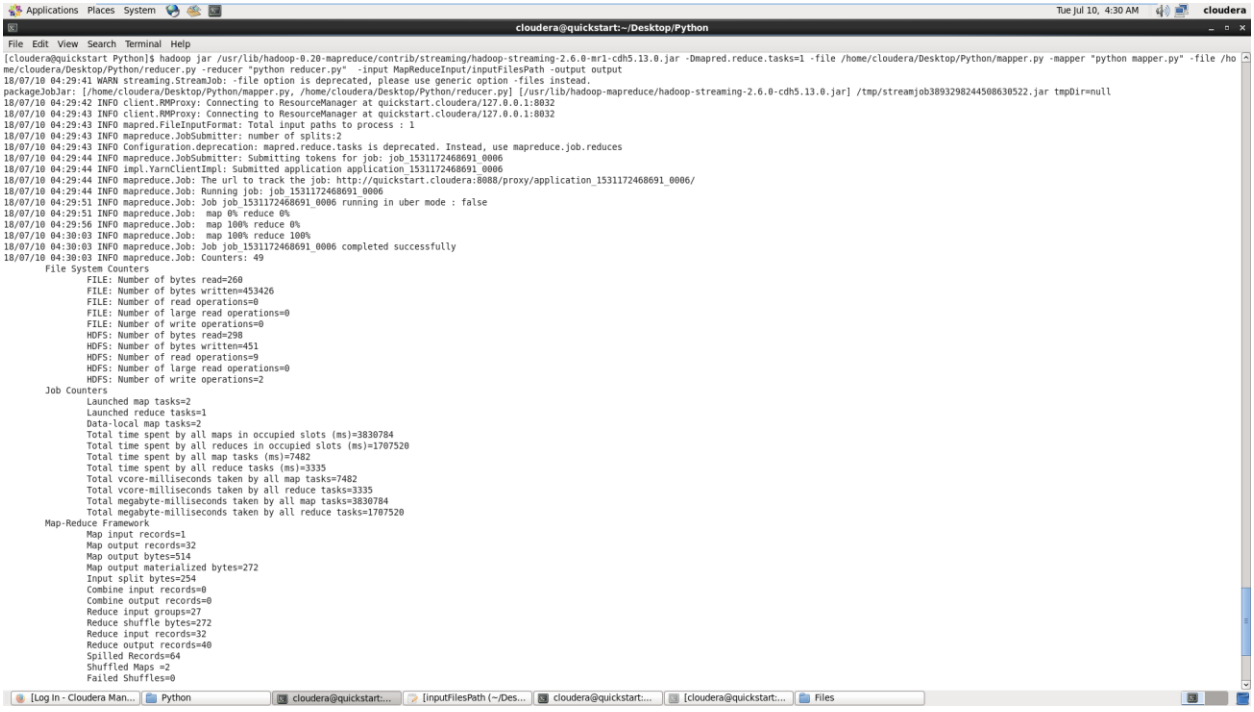
inputFilesPath



Command:



Command Running:



Output:

```
[cloudera@quickstart Python]$ hadoop fs -cat output/*0000
- file1.txt file2.txt
after file1.txt
and file1.txt file2.txt
based file1.txt
file file1.txt
has file2.txt
hey file1.txt file2.txt
is file1.txt file2.txt
map file2.txt
methods file1.txt
name file1.txt file2.txt
on file1.txt
output file1.txt file2.txt
project file2.txt
python file2.txt
reduce file1.txt file2.txt
the file1.txt
two file2.txt
upon file2.txt
word file2.txt
[cloudera@quickstart Python]$
```

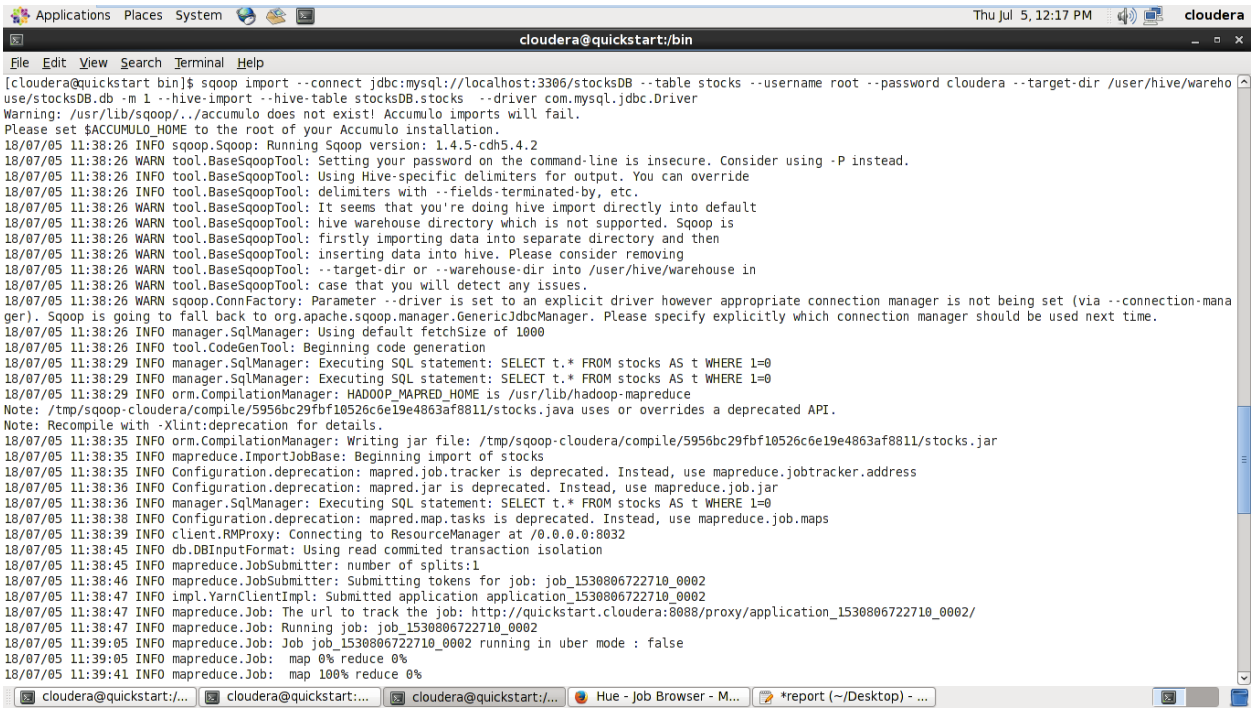
3.2 Sqoop Task: Loading Data from RDBMS to HDFS

Dataset used : stocks.csv

Task – to load data from RDBMS (MySQL) to HDFS & then load data from HDFS to Hive tables

Step 1 – Loading data from RDBMS (MySQL) to HDFS

Command : sqoop import --connect jdbc:mysql://localhost:3306/stocksDB --table stocks --username root --password cloudera --target-dir /user/hive/warehouse/stocksDB.db -m 1 --hive-import --hive-table stocksDB.stocks --driver com.mysql.jdbc.Driver



3.3 Stocks Analysis using Hive

Dataset used : Stocks.csv

Step 1 - Opening Hive2 or Thrift Server & make a connection with it.

```
[cloudera@quickstart Desktop]$ cd $HIVE_HOME/bin
[cloudera@quickstart bin]$ beeline
Beeline version 1.1.0-cdh5.13.0 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default
scan complete in 2ms
Connecting to jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: cloudera
Enter password for jdbc:hive2://localhost:10000/default: ********
Connected to: Apache Hive (version 1.1.0-cdh5.13.0)
Driver: Hive JDBC (version 1.1.0-cdh5.13.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000/default>
```

Step 2 - Creating & using Database stocks_DB

Command - create database stocks_DB;
use stocks_DB;

```
0: jdbc:hive2://localhost:10000/default> create database stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a): create database stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a); Time taken: 0.016 seconds
INFO  : Executing command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a): create database stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a); Time taken: 0.459 seconds
INFO  : OK
No rows affected (0.503 seconds)
0: jdbc:hive2://localhost:10000/default> use stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595): use stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595); Time taken: 0.021 seconds
INFO  : Executing command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595): use stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595); Time taken: 0.014 seconds
INFO  : OK
No rows affected (0.055 seconds)
```

Step 3 - Creating External table stocks

Command –

```
create external table if not exists stocks
(
exch STRING,
symbol STRING,
date STRING,
```

```
stocks_price_open FLOAT,
stocks_price_high FLOAT,
stocks_price_low FLOAT,
stocks_price_close FLOAT,
volume INT,
price_adj_close FLOAT
)
COMMENT 'This is External Table for Stocks'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY',';
```

```
0: jdbc:hive2://localhost:10000/default> create external table if not exists stocks
. . . . . .> (
. . . . . .> exch STRING,
. . . . . .> symbol STRING,
. . . . . .> date STRING,
. . . . . .> stocks_price_open FLOAT,
. . . . . .> stocks_price_high FLOAT,
. . . . . .> stocks_price_low FLOAT,
. . . . . .> stocks_price_close FLOAT,
. . . . . .> volume INT,
. . . . . .> price_adj_close FLOAT
. . . . . .> )
. . . . . .> COMMENT 'This is External Table for Stocks'
. . . . . .> ROW FORMAT DELIMITED
. . . . . .> FIELDS TERMINATED BY',';
```

```
0: jdbc:hive2://localhost:10000/default> show tables in stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade): show tables in stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade); Time taken: 0.019 seconds
INFO  : Executing command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade): show tables in stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade); Time taken: 0.036 seconds
INFO  : OK
+-----------+--+
| tab_name  |
+-----------+--+
| stocks    |
+-----------+--+
```

Step 4 - Executing Queries

Query 1: Finding top 10 companies according to number of stocks

Command - select symbol,count(symbol) as count from stocks group by symbol order by count desc limit 10;

```
0: jdbc:hive2://localhost:10000/default> select symbol,count(symbol) as count from stocks group by symbol order by count desc limit 10;
INFO  : Compiling command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482): select symbol,count(symbol) as count from stocks group by symbol
 order by count desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:symbol, type:string, comment:null), FieldSchema(name:count, type:bigint, comment:null)],
 properties:null)
INFO  : Completed compiling command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482); Time taken: 0.273 seconds
INFO  : Executing command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482): select symbol,count(symbol) as count from stocks group by symbol
 order by count desc limit 10
INFO  : Query ID = hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0006
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0006/
INFO  : Starting Job = job_1531159788380_0006, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0006/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0006
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:23:41,179 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:23:48,588 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.92 sec
INFO  : 2018-07-10 00:23:55,938 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.6 sec
INFO  : MapReduce Total cumulative CPU time: 4 seconds 600 msec
INFO  : Ended Job = job_1531159788380_0006
INFO  : Launching Job 2 out of 2
INFO  : Starting task [Stage-2:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
```

```
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0007
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0007/
INFO  : Starting Job = job_1531159788380_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0007/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0007
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:24:04,053 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:24:09,438 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.13 sec
INFO  : 2018-07-10 00:24:16,793 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.79 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 790 msec
INFO  : Ended Job = job_1531159788380_0007
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.6 sec   HDFS Read: 40998932 HDFS Write: 4994 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.79 sec   HDFS Read: 10050 HDFS Write: 91 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 390 msec
INFO  : Completed executing command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482); Time taken: 43.735 seconds
INFO  : OK
+---------+--------+--+
| symbol  | count  |
+---------+--------+--+
| AA      | 12109  |
| AEP     | 10121  |
| AET     | 8353   |
| AXP     | 8290   |
| ASA     | 8095   |
| AMR     | 7590   |
| APA     | 7091   |
| AVP     | 7085   |
| AVT     | 7084   |
| ALK     | 6831   |
+---------+--------+--+
10 rows selected (44.14 seconds)
```

Query 2: Finding top 10 years according to average volume of stocks

Command - select year(date),avg(volume) as avgVol from stocks group by year(date) order by avgVol desc limit 10;

```
0: jdbc:hive2://localhost:10000/default> select year(date),avg(volume) as avgVol from stocks group by year(date) order by avgVol desc limit 10;
INFO  : Compiling command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2): select year(date),avg(volume) as avgVol from stocks group by yea
r(date) order by avgVol desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:_c0, type:int, comment:null), FieldSchema(name:avgvol, type:double, comment:null)], prop
erties:null)
INFO  : Completed compiling command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2); Time taken: 0.107 seconds
INFO  : Executing command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2): select year(date),avg(volume) as avgVol from stocks group by yea
r(date) order by avgVol desc limit 10
INFO  : Query ID = hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0008
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0008/
INFO  : Starting Job = job_1531159788380_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0008/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0008
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:26:43,971 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:26:52,518 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.64 sec
INFO  : 2018-07-10 00:26:58,831 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.37 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 370 msec
INFO  : Ended Job = job_1531159788380_0008
INFO  : Launching Job 2 out of 2
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0009
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:27:05,691 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:27:12,189 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.63 sec
INFO  : 2018-07-10 00:27:18,535 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.34 sec
INFO  : MapReduce Total cumulative CPU time: 3 seconds 340 msec
INFO  : Ended Job = job_1531159788380_0009
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.37 sec   HDFS Read: 40999849 HDFS Write: 1468 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.34 sec   HDFS Read: 6538 HDFS Write: 235 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 10 seconds 710 msec
INFO  : Completed executing command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2); Time taken: 43.49 seconds
INFO  : OK
+-------+---------------------+--+
|  _c0  |       avgvol        |
+-------+---------------------+--+
| 2009  | 2144999.0106877508  |
| 2010  | 2009298.4872611465  |
| 2008  | 1941244.2091383133  |
| 2007  | 1337635.7611070648  |
| 1983  | 1087647.1091521909  |
| 2006  | 1059718.7874033398  |
| 1982  | 1021882.4769433466  |
| 2003  | 941810.9683610097   |
| 2005  | 902292.056767634    |
| 2002  | 899578.376947301    |
+-------+---------------------+--+
10 rows selected (43.678 seconds)
```

Query 3: Finding top 10 month & year according to rise in stocks price

Command - select month(date) as month ,year(date) as year, (max(stocks_price_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10;

```
10 rows selected (45.078 seconds)
0: jdbc:hive2://localhost:10000/default> select month(date) as month ,year(date) as year, (max(stocks_price_high) - min(stocks_price_low)) as stocksPriceRise
 from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10;
INFO  : Compiling command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137): select month(date) as month ,year(date) as year, (max(stocks_pri
ce_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:month, type:int, comment:null), FieldSchema(name:year, type:int, comment:null), FieldSch
ema(name:stockspricerise, type:float, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137); Time taken: 0.122 seconds
INFO  : Executing command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137): select month(date) as month ,year(date) as year, (max(stocks_pri
ce_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10
INFO  : Query ID = hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0010
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0010/
INFO  : Starting Job = job_1531159788380_0010, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0010/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0010
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:31:31,119 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:31:41,939 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.48 sec
INFO  : 2018-07-10 00:31:49,259 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.72 sec
INFO  : MapReduce Total cumulative CPU time: 9 seconds 720 msec
INFO  : Ended Job = job_1531159788380_0010
INFO  : Launching Job 2 out of 2
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0011
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0011/
INFO  : Starting Job = job_1531159788380_0011, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0011/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0011
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:31:56,033 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:32:02,323 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.18 sec
INFO  : 2018-07-10 00:32:09,626 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.88 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 880 msec
INFO  : Ended Job = job_1531159788380_0011
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.72 sec   HDFS Read: 41001141 HDFS Write: 14686 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.88 sec   HDFS Read: 19845 HDFS Write: 147 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 12 seconds 600 msec
INFO  : Completed executing command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137); Time taken: 46.419 seconds
INFO  : OK
+--------+-------+---------------------+--+
| month  | year  |   stockspricerise   |
+--------+-------+---------------------+--+
| 2      | 2007  | 465.7699890136719   |
| 12     | 2006  | 444.5899963378906   |
| 1      | 2007  | 444.5199890136719   |
| 6      | 2007  | 435.6399841308594   |
| 9      | 2008  | 430.4900207519531   |
| 5      | 2007  | 429.2699890136719   |
| 3      | 2007  | 420.6000061035156   |
| 7      | 2007  | 420.42999267578125  |
| 10     | 2007  | 418.9100036621094   |
| 4      | 2007  | 415.760009765625    |
+--------+-------+---------------------+--+
10 rows selected (46.618 seconds)
```

Query 4: Finding top 10 max difference between stock price close & stock price adjustment close

Command - select price_adj_close-stocks_price_close as adj from stocks order by adj desc limit 10;

```
0: jdbc:hive2://localhost:10000/default> select price_adj_close-stocks_price_close as adj from stocks order by adj  desc limit 10;
INFO  : Compiling command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595): select price_adj_close-stocks_price_close as adj from stocks ord
er by adj  desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:adj, type:float, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595); Time taken: 0.101 seconds
INFO  : Executing command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595): select price_adj_close-stocks_price_close as adj from stocks ord
er by adj  desc limit 10
INFO  : Query ID = hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0013
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0013/
INFO  : Starting Job = job_1531159788380_0013, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0013/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0013
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:38:24,570 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:38:33,894 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.97 sec
INFO  : 2018-07-10 00:38:40,247 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.66 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 660 msec
INFO  : Ended Job = job_1531159788380_0013
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.66 sec   HDFS Read: 40999660 HDFS Write: 81 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 660 msec
INFO  : Completed executing command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595); Time taken: 22.513 seconds
INFO  : OK
```

```
+----------------------+--+
|          adj         |
+----------------------+--+
| 1843.7900390625      |
| 1810.360107421875    |
| 1800.4000244140625   |
| 1786.3199462890625   |
| 1780.81005859375     |
| 1779.239990234375    |
| 1779.239990234375    |
| 1769.280029296875    |
| 1769.280029296875    |
| 1766.22998046875     |
+----------------------+--+
10 rows selected (22.688 seconds)
```

Query 5: Finding top 10 companies according to volume of stocks

Command - select symbol,max(volume) as MAXVOL from stocks group by symbol order by MAXVOL desc limit 10;

```
0: jdbc:hive2://localhost:10000/default> select symbol,max(volume) as MAXVOL from stocks group by symbol order by MAXVOL desc limit 10;
INFO  : Compiling command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf): select symbol,max(volume) as MAXVOL from stocks group by symbol
order by MAXVOL desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:symbol, type:string, comment:null), FieldSchema(name:maxvol, type:int, comment:null)], p
roperties:null)
INFO  : Completed compiling command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf); Time taken: 0.094 seconds
INFO  : Executing command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf): select symbol,max(volume) as MAXVOL from stocks group by symbol
order by MAXVOL desc limit 10
INFO  : Query ID = hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0014
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0014/
INFO  : Starting Job = job_1531159788380_0014, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0014/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0014
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:40:39,328 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:40:46,659 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.52 sec
INFO  : 2018-07-10 00:40:52,928 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.1 sec
INFO  : MapReduce Total cumulative CPU time: 5 seconds 100 msec
INFO  : Ended Job = job_1531159788380_0014
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0015
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0015/
INFO  : Starting Job = job_1531159788380_0015, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0015/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0015
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:40:59,769 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:41:06,043 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.13 sec
INFO  : 2018-07-10 00:41:12,626 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.87 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 870 msec
INFO  : Ended Job = job_1531159788380_0015
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.1 sec   HDFS Read: 40999017 HDFS Write: 5259 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.87 sec   HDFS Read: 10299 HDFS Write: 135 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 970 msec
INFO  : Completed executing command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf); Time taken: 40.643 seconds
INFO  : OK
+---------+------------+--+
| symbol  |   maxvol   |
+---------+------------+--+
| AVP     | 318182200  |
| AA      | 242106500  |
| AMD     | 163101700  |
| AIG     | 148878600  |
| ABK     | 112834200  |
| AXL     | 103526300  |
| AXP     | 90336900   |
| AMR     | 89947900   |
| ACE     | 74437100   |
| AET     | 67642400   |
+---------+------------+--+
10 rows selected (40.809 seconds)
```

Query 6: Finding top 10 times when price adjustment close and price close has positive difference

Command - select count(price_adj_close-stocks_price_close) as adj from stocks where (price_adj_close-stocks_price_close) > 0 limit 10;

```
0: jdbc:hive2://localhost:10000/default> select count(price_adj_close-stocks_price_close) as adj from stocks where (price_adj_close-stocks_price_close) !0 li
mit 10;
```

```
INFO  : 2018-07-10 01:47:37,214 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.98 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 980 msec
INFO  : Ended Job = job_1531165075887_0003
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.98 sec   HDFS Read: 81992182 HDFS Write: 8 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 980 msec
INFO  : Completed executing command(queryId=hive_20180710014646_62180fb6-28ce-4a2a-b00a-fa1b14032a83); Time taken: 38.261 seconds
INFO  : OK
+----------+--+
|   adj    |
+----------+--+
| 1284850  |
+----------+--+
```

Query 7: Finding top 10 dates according to number of stocks of company AVP

Command - select date, volume from stocks where symbol = "AVP" order by volume desc limit 10;

```
0: jdbc:hive2://localhost:10000/default> select date, volume from stocks where symbol = "AVP" order by volume desc limit 10;
INFO  : Compiling command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8): select date, volume from stocks where symbol = "AVP" order by vo
lume desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:date, type:string, comment:null), FieldSchema(name:volume, type:int, comment:null)], pro
perties:null)
INFO  : Completed compiling command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8); Time taken: 0.144 seconds
INFO  : Executing command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8): select date, volume from stocks where symbol = "AVP" order by vo
lume desc limit 10
INFO  : Query ID = hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0016
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0016/
INFO  : Starting Job = job_1531159788380_0016, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0016/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0016
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:43:42,249 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:43:49,555 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.74 sec
INFO  : 2018-07-10 00:43:56,876 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.53 sec
INFO  : MapReduce Total cumulative CPU time: 5 seconds 530 msec
INFO  : Ended Job = job_1531159788380_0016
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.53 sec   HDFS Read: 40999895 HDFS Write: 202 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 5 seconds 530 msec
INFO  : Completed executing command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8); Time taken: 22.903 seconds
INFO  : OK
+--------------+-------------+--+
|     date     |   volume    |
+--------------+-------------+--+
| 1988-05-12   | 318182200   |
| 1988-02-12   | 165642600   |
| 1991-03-14   | 93343200    |
| 1987-11-13   | 87786600    |
| 1988-02-08   | 82144000    |
| 1989-05-03   | 79389800    |
| 1988-02-09   | 77893300    |
| 1989-05-10   | 57950900    |
| 1989-05-18   | 46672000    |
| 1989-05-19   | 38488800    |
+--------------+-------------+--+
10 rows selected (23.111 seconds)
```

Step 3 - Creating External table stocks

Command –

```
create table stocks_partition(
exch STRING,
symbol STRING,
date STRING,
stocks_price_close FLOAT,
stocks_price_high FLOAT,
stocks_price_low FLOAT,
stocks_price_open FLOAT,
volume INT,
price_adj_close FLOAT)
PARTITIONED BY (exch_name STRING,yr STRING, sym STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

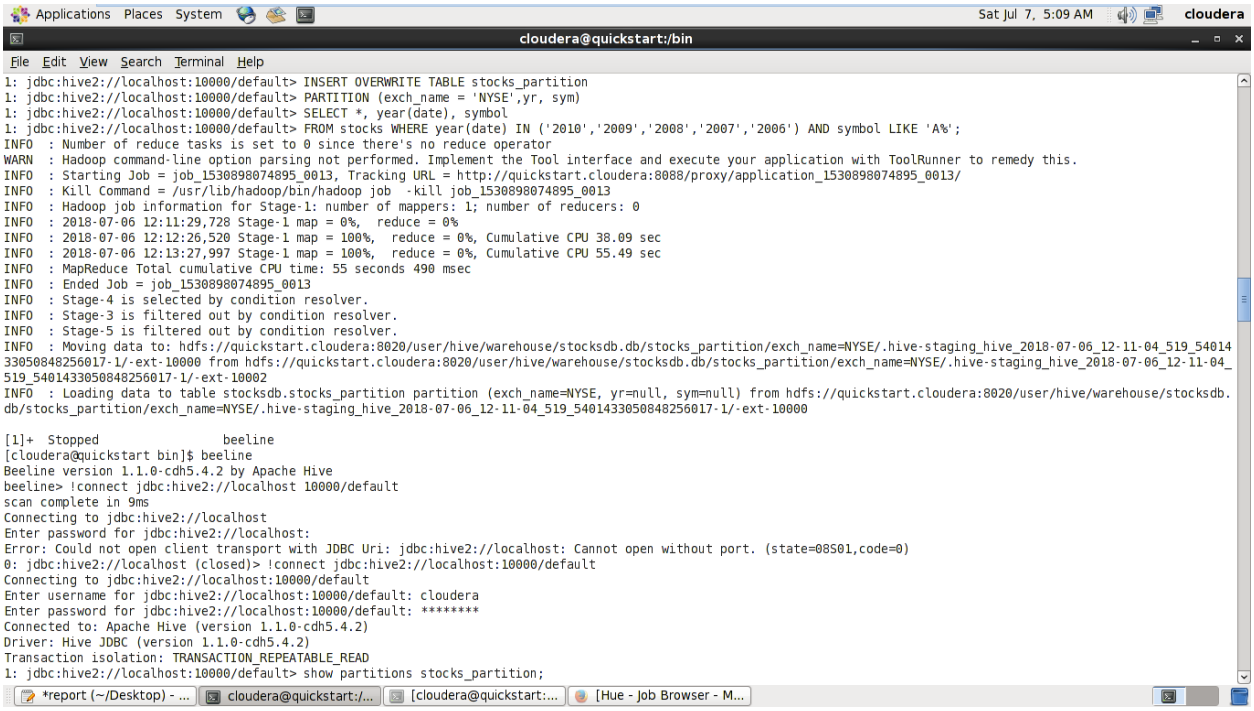
```
0: jdbc:hive2://localhost:10000/default> create table stocks_partition(
. . . . . .> exch STRING,
. . . . . .> symbol STRING,
. . . . . .> date STRING,
. . . . . .> stocks_price_close FLOAT,
. . . . . .> stocks_price_high FLOAT,
. . . . . .> stocks_price_low FLOAT,
. . . . . .> stocks_price_open FLOAT,
. . . . . .> volume INT,
. . . . . .> price_adj_close FLOAT)
. . . . . .> PARTITIONED BY (exch_name STRING,yr STRING, sym STRING)
. . . . . .> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

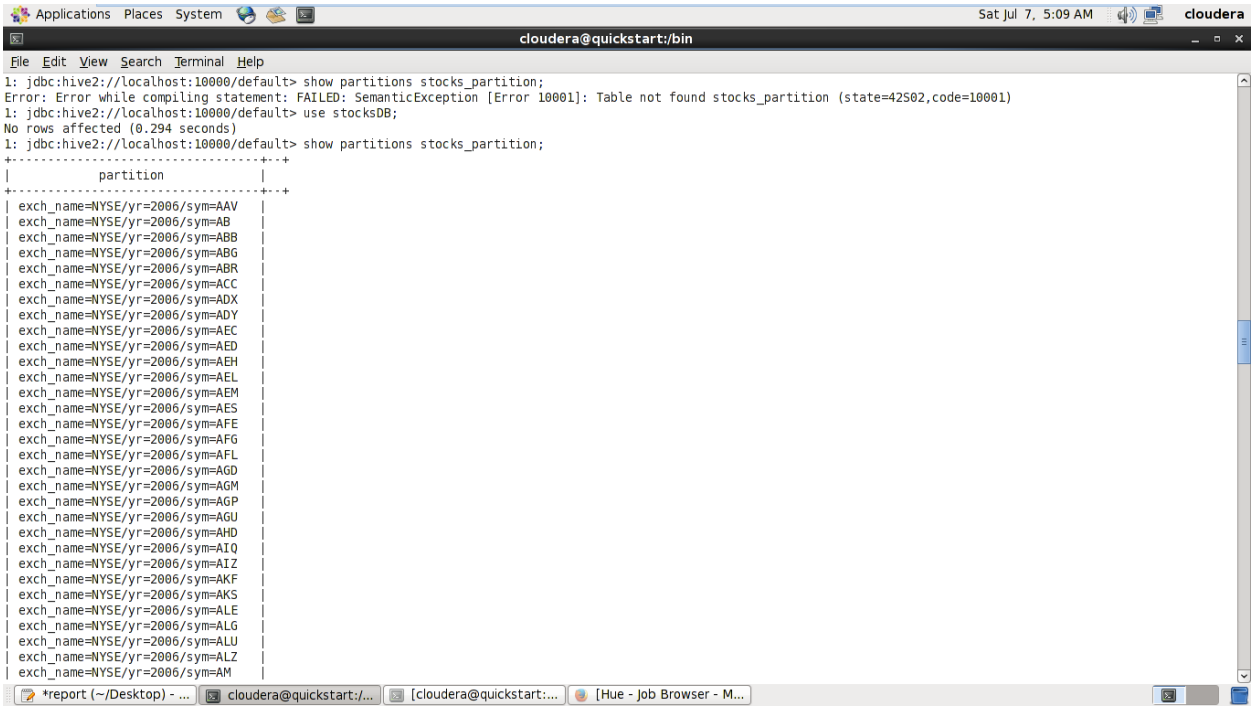
```
0: jdbc:hive2://localhost:10000/default> set hive.exec.dynamic.partition = true;
No rows affected (0.013 seconds)
0: jdbc:hive2://localhost:10000/default> set hive.exec.max.dynamic.partitions = 2000;
No rows affected (0.005 seconds)
0: jdbc:hive2://localhost:10000/default> set hive.exec.max.dynamic.partitions.pernode = 1000;
No rows affected (0.004 seconds)
```

Query 8: Loading data into partitions

Command –

```
INSERT OVERWRITE TABLE stocks_partition
PARTITION (exch_name = 'NYSE',yr, sym)
SELECT *, year(date), symbol
FROM stocks WHERE year(date) IN ('2010','2009','2008','2007','2006') AND symbol LIKE
'A%';
```


Applications Places System
Sat Jul 7, 5:09 AM
cloudera
cloudera@quickstart:/bin
File Edit View Search Terminal Help
1: jdbc:hive2://localhost:10000/default> INSERT OVERWRITE TABLE stocks_partition
1: jdbc:hive2://localhost:10000/default> PARTITION (exch_name = 'NYSE',yr, sym)
1: jdbc:hive2://localhost:10000/default> SELECT *, year(date), symbol
1: jdbc:hive2://localhost:10000/default> FROM stocks WHERE year(date) IN ('2010','2009','2008','2007','2006') AND symbol LIKE 'A%';
INFO : Number of reduce tasks is set to 0 since there's no reduce operator
WARN : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO : Starting Job = job_1530898074895_0013, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0013/
INFO : Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1530898074895_0013
INFO : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
INFO : 2018-07-06 12:11:29,728 Stage-1 map = 0%, reduce = 0%
INFO : 2018-07-06 12:12:26,520 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 38.09 sec
INFO : 2018-07-06 12:13:27,997 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 55.49 sec
INFO : MapReduce Total cumulative CPU time: 55 seconds 490 msec
INFO : Ended Job = job_1530898074895_0013
INFO : Stage-4 is selected by condition resolver.
INFO : Stage-3 is filtered out by condition resolver.
INFO : Stage-5 is filtered out by condition resolver.
INFO : Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_519_5401433050848256017-1/-ext-10000 from hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_519_5401433050848256017-1/-ext-10002
INFO : Loading data to table stocksdb.stocks_partition partition (exch_name=NYSE, yr=null, sym=null) from hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_519_5401433050848256017-1/-ext-10000
[1]+ Stopped beeline
[cloudera@quickstart bin]$ beeline
Beeline version 1.1.0-cdh5.4.2 by Apache Hive
beeline> !connect jdbc:hive2://localhost 10000/default
scan complete in 9ms
Connecting to jdbc:hive2://localhost
Enter password for jdbc:hive2://localhost:
Error: Could not open client transport with JDBC Uri: jdbc:hive2://localhost: Cannot open without port. (state=08S01,code=0)
0: jdbc:hive2://localhost (closed)> !connect jdbc:hive2://localhost:10000/default
Connecting to jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: cloudera
Enter password for jdbc:hive2://localhost:10000/default: ********
Connected to: Apache Hive (version 1.1.0-cdh5.4.2)
Driver: Hive JDBC (version 1.1.0-cdh5.4.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
*report (~/Desktop) - ...
cloudera@quickstart:/...
[cloudera@quickstart:...
[Hue - Job Browser - M...


Applications Places System
Sat Jul 7, 5:09 AM
cloudera
cloudera@quickstart:/bin
File Edit View Search Terminal Help
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
Error: Error while compiling statement: FAILED: SemanticException [Error 10001]: Table not found stocks_partition (state=42S02,code=10001)
1: jdbc:hive2://localhost:10000/default> use stocksDB;
No rows affected (0.294 seconds)
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
partition
exch_name=NYSE/yr=2006/sym=AAV
exch_name=NYSE/yr=2006/sym=AB
exch_name=NYSE/yr=2006/sym=ABB
exch_name=NYSE/yr=2006/sym=ABG
exch_name=NYSE/yr=2006/sym=ABR
exch_name=NYSE/yr=2006/sym=ACC
exch_name=NYSE/yr=2006/sym=ADX
exch_name=NYSE/yr=2006/sym=ADY
exch_name=NYSE/yr=2006/sym=AEC
exch_name=NYSE/yr=2006/sym=AED
exch_name=NYSE/yr=2006/sym=AEH
exch_name=NYSE/yr=2006/sym=AEL
exch_name=NYSE/yr=2006/sym=AEM
exch_name=NYSE/yr=2006/sym=AES
exch_name=NYSE/yr=2006/sym=AFE
exch_name=NYSE/yr=2006/sym=AFG
exch_name=NYSE/yr=2006/sym=AFL
exch_name=NYSE/yr=2006/sym=AGD
exch_name=NYSE/yr=2006/sym=AGM
exch_name=NYSE/yr=2006/sym=AGP
exch_name=NYSE/yr=2006/sym=AGU
exch_name=NYSE/yr=2006/sym=AHD
exch_name=NYSE/yr=2006/sym=AIQ
exch_name=NYSE/yr=2006/sym=AIZ
exch_name=NYSE/yr=2006/sym=AKF
exch_name=NYSE/yr=2006/sym=AKS
exch_name=NYSE/yr=2006/sym=ALE
exch_name=NYSE/yr=2006/sym=ALG
exch_name=NYSE/yr=2006/sym=ALU
exch_name=NYSE/yr=2006/sym=ALZ
exch_name=NYSE/yr=2006/sym=AM
*report (~/Desktop) - ...
cloudera@quickstart:/...
[cloudera@quickstart:...
[Hue - Job Browser - M...

Query 9: Finding maximum stock price in AVP company partition

Command –

SELECT MAX(stocks_price_high) as max_high FROM stocks_partition
WHERE sym = 'AVP';

```
1: jdbc:hive2://localhost:10000/default> SELECT sym,MAX(stocks_price_high) FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
Error: Error while compiling statement: FAILED: SemanticException [Error 10025]: Line 1:7 Expression not in GROUP BY key 'sym' (state=42000,code=10025)
1: jdbc:hive2://localhost:10000/default> SELECT MAX(stocks_price_high) FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
WARN  : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO  : Starting Job = job_1530898074895_0020, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0020/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1530898074895_0020
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-07 11:58:48,034 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-07 11:59:00,248 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.72 sec
INFO  : 2018-07-07 11:59:13,213 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 8.63 sec
INFO  : MapReduce Total cumulative CPU time: 8 seconds 630 msec
INFO  : Ended Job = job_1530898074895_0020
+--------------------+--+
|        _c0         |
+--------------------+--+
| 45.34000015258789  |
+--------------------+--+
```

Query 10: Finding minimum stock price in AVP company partition

Command –
SELECT MIN(stocks_price_low) as min_low FROM stocks_partition
WHERE sym = 'AVP';

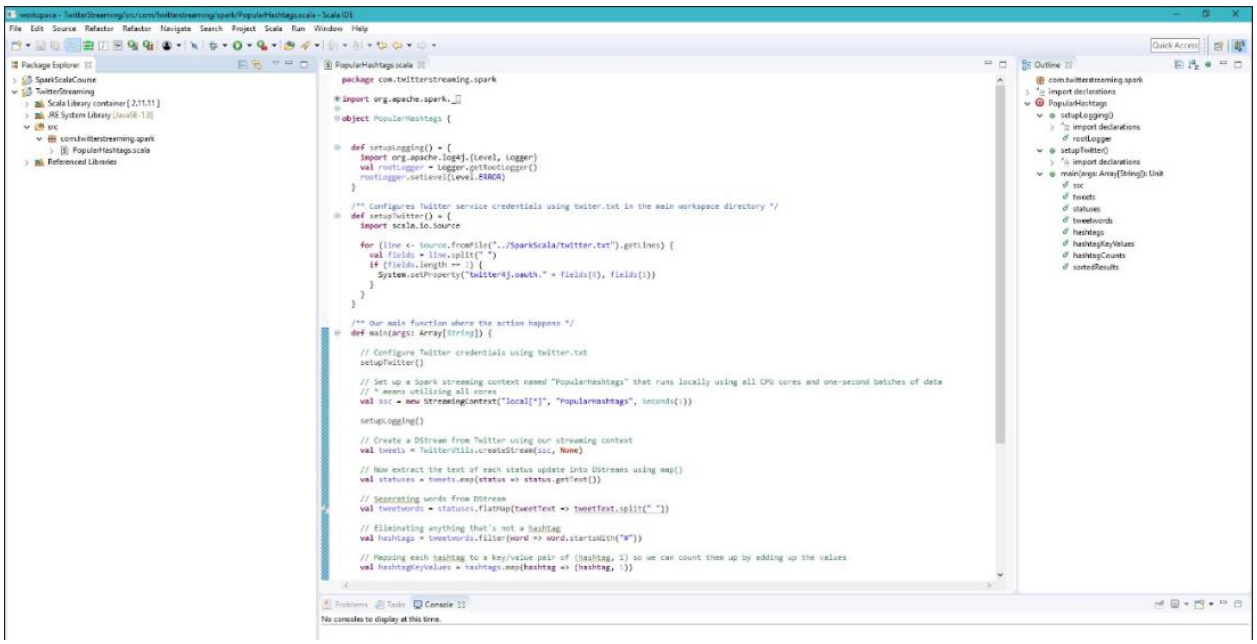
```
1: jdbc:hive2://localhost:10000/default> SELECT MIN(stocks_price_low) as min_low FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
WARN  : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO  : Starting Job = job_1530898074895_0022, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0022/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1530898074895_0022
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-07 12:01:58,545 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-07 12:02:10,499 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.6 sec
INFO  : 2018-07-07 12:02:22,458 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 8.0 sec
INFO  : MapReduce Total cumulative CPU time: 8 seconds 0 msec
INFO  : Ended Job = job_1530898074895_0022
+--------------------+--+
|       min_low      |
+--------------------+--+
| 14.399999618530273 |
+--------------------+--+
```

3.4 Twitter's Top 10 popular Hashtag Streaming per second using Apache Spark

Technologies used –

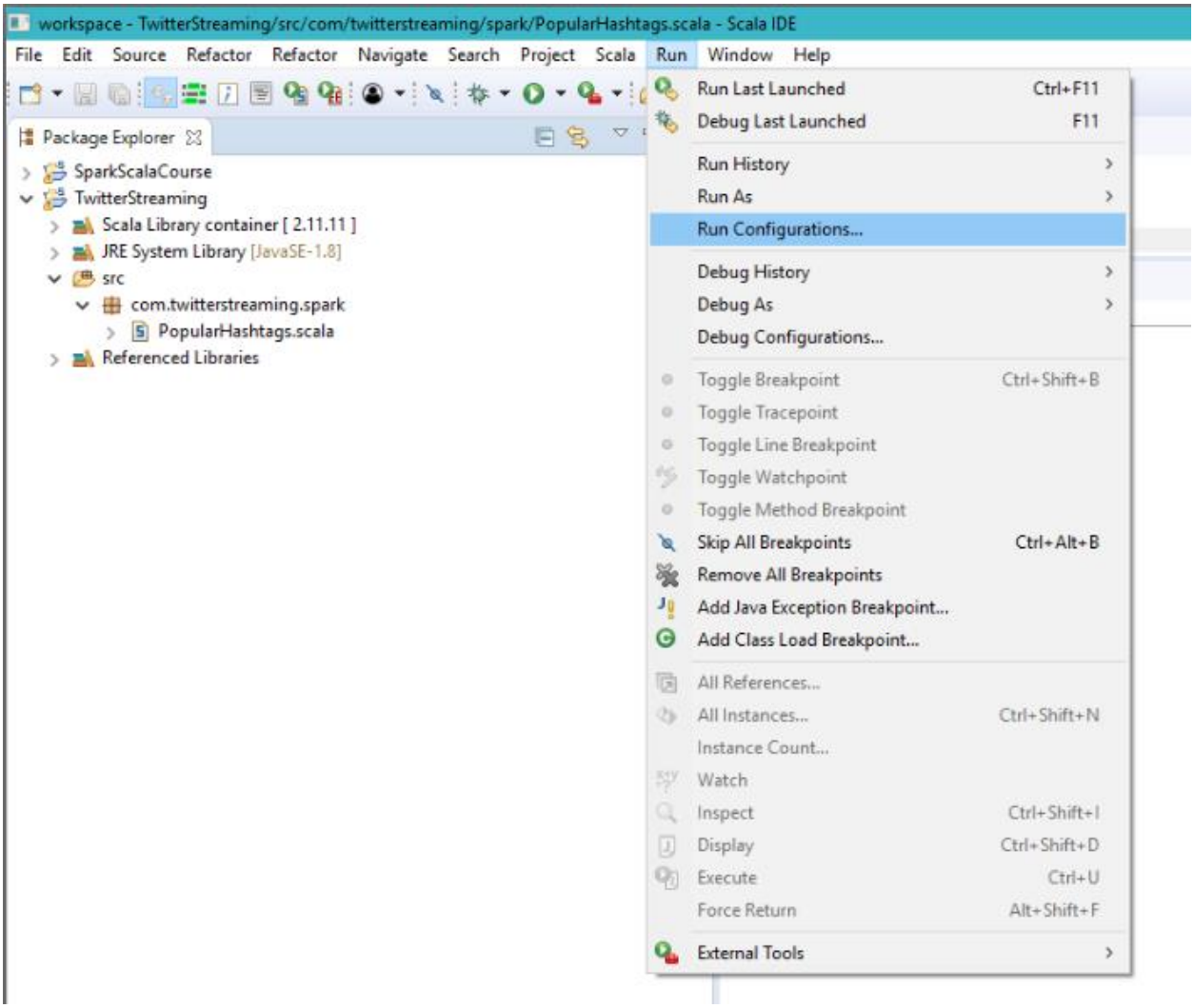
- Spark
- Scala 2.11.11
- Libraries used : dsstream-twitter_2.11-0.1.0-SNAPSHOT, twitter4j-core-4.0.4, twitter4j-stream-4.0.4
- Eclipse Scala IDE

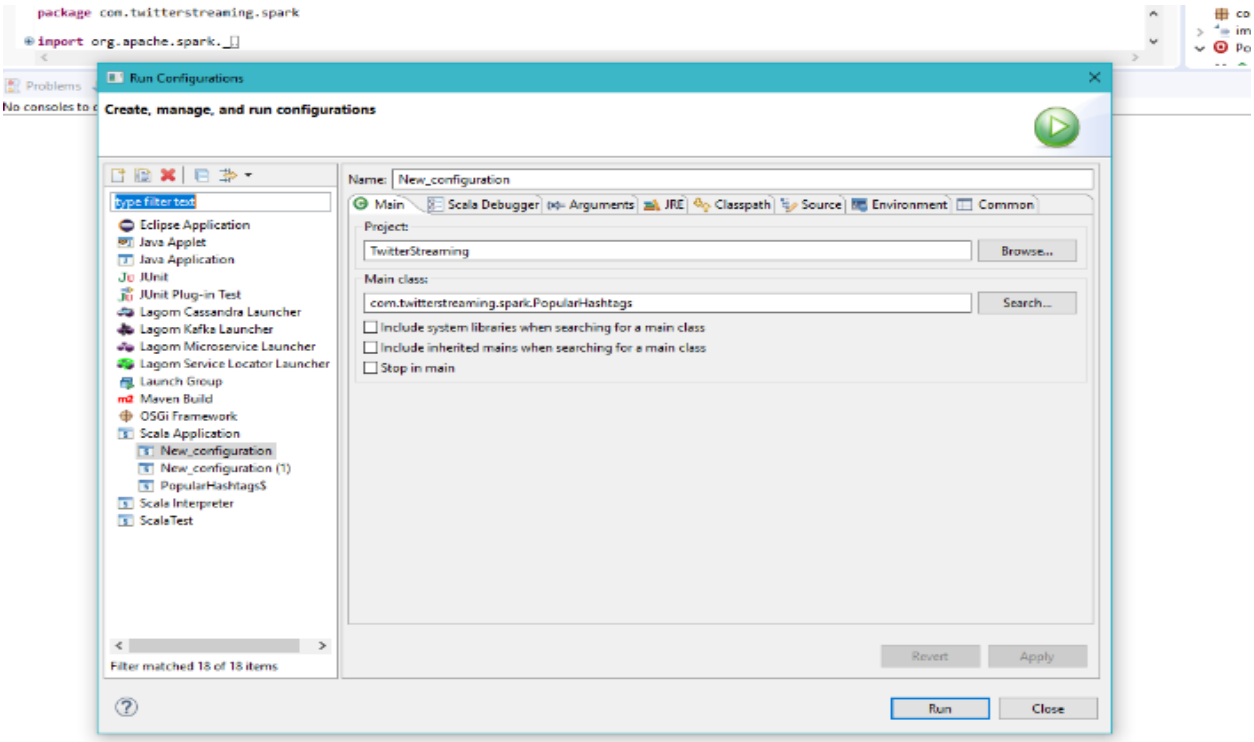
Scala Code:



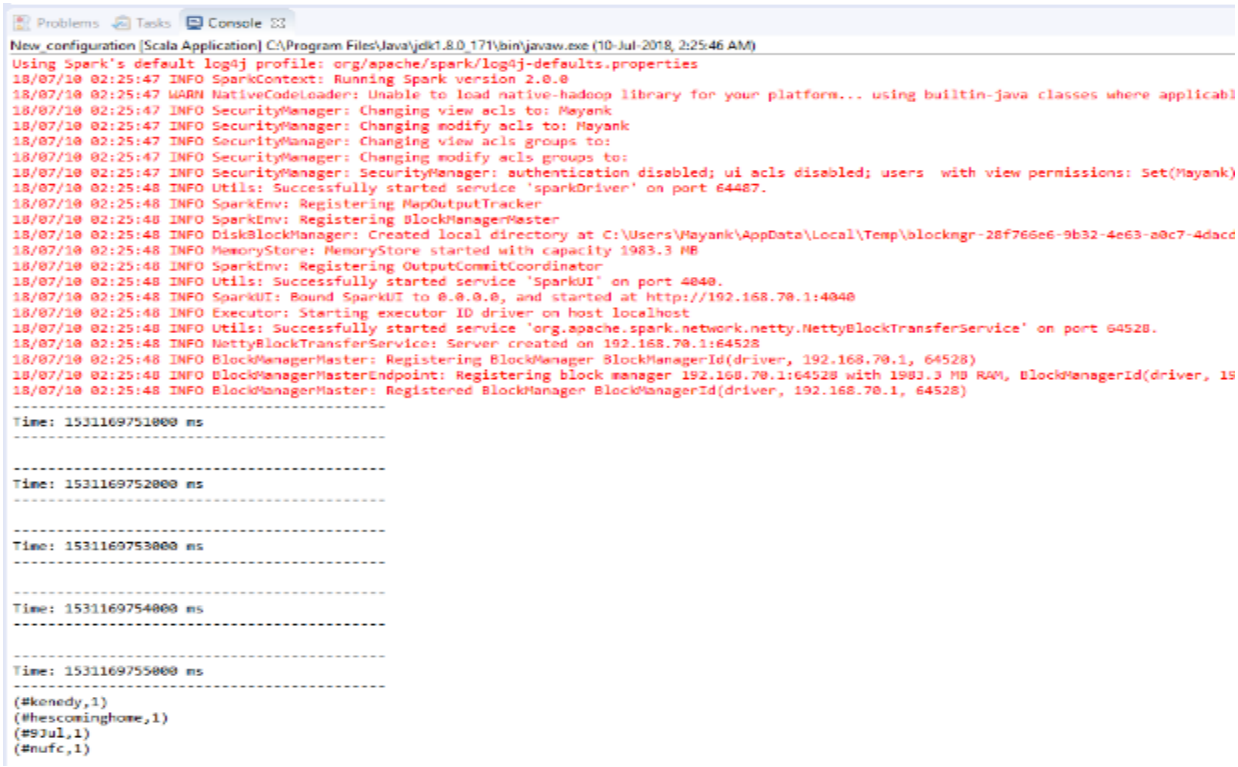
```
package com.twitterstreaming.spark

import org.apache.spark._

object PopularHashtags {

  def setupLogging() = {
    import org.apache.log4j.{Level, Logger}
    val rootLogger = Logger.getRootLogger()
    rootLogger.setLevel(Level.ERROR)
  }

  /** Configures Twitter service credentials using twitter.txt in the main workspace directory */
  def setupTwitter() = {
    import scala.io.Source

    for (line <- Source.fromFile("../SparkScala/twitter.txt").getLines) {
      val fields = line.split(" ")
      if (fields.length == 2) {
        System.setProperty("twitter4j.oauth." + fields(0), fields(1))
      }
    }
  }

  /** Our main function where the action happens */
  def main(args: Array[String]) {

    // Configure Twitter credentials using twitter.txt
    setupTwitter()

    // Set up a Spark streaming context named "PopularHashtags" that runs locally using all CPU cores and one-second batches of data
    // * means utilizing all cores
    val ssc = new StreamingContext("local[*]", "PopularHashtags", Seconds(1))

    setupLogging()

    // Create a DStream from Twitter using our streaming context
    val tweets = TwitterUtils.createStream(ssc, None)

    // Now extract the text of each status update into DStreams using map()
    val statuses = tweets.map(status => status.getText())

    // Separating words from DStream
    val tweetwords = statuses.flatMap(tweetText => tweetText.split(" "))

    // Eliminating anything that's not a hashtag
    val hashtags = tweetwords.filter(word => word.startsWith("#"))

    // Mapping each hashtag to a key/value pair of (hashtag, 1) so we can count them up by adding up the values
    val hashtagKeyValues = hashtags.map(hashtag => (hashtag, 1))
```

Running Script:



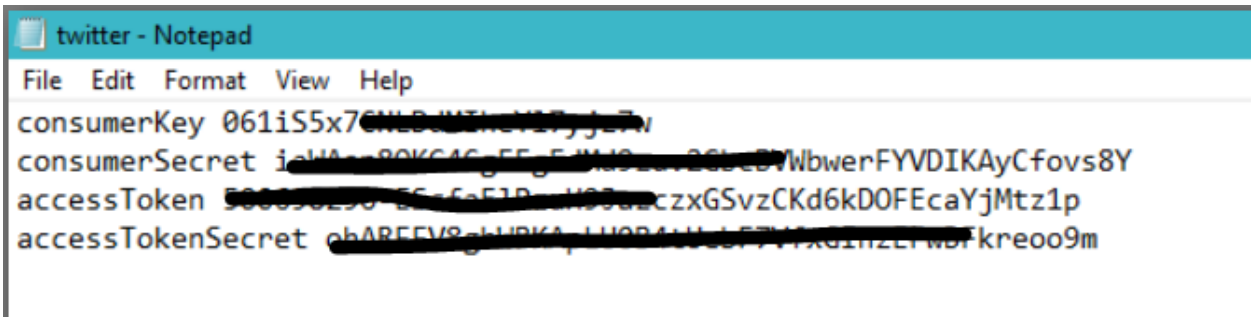

package com.twitterstreaming.spark
import org.apache.spark._
Run Configurations
Create, manage, and run configurations
Name: New_configuration
Main
Scala Debugger
Arguments
JRE
Classpath
Source
Environment
Common
Project:
TwitterStreaming
Browse...
Main class:
com.twitterstreaming.spark.PopularHashtags
Search...
Include system libraries when searching for a main class
Include inherited mains when searching for a main class
Stop in main
Eclipse Application
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Lagom Cassandra Launcher
Lagom Kafka Launcher
Lagom Microservice Launcher
Lagom Service Locator Launcher
Launch Group
Maven Build
OSGi Framework
Scala Application
New_configuration
New_configuration (1)
PopularHashtags$
Scala Interpreter
ScalaTest
Filter matched 18 of 18 items
Revert
Apply
Run
Close

Output:


Problems
Tasks
Console
New_configuration [Scala Application] C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe (10-Jul-2018, 2:25:46 AM)
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/07/10 02:25:47 INFO SparkContext: Running Spark version 2.0.0
18/07/10 02:25:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicabl
18/07/10 02:25:47 INFO SecurityManager: Changing view acls to: Mayank
18/07/10 02:25:47 INFO SecurityManager: Changing modify acls to: Mayank
18/07/10 02:25:47 INFO SecurityManager: Changing view acls groups to:
18/07/10 02:25:47 INFO SecurityManager: Changing modify acls groups to:
18/07/10 02:25:47 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(Mayank)
18/07/10 02:25:48 INFO Utils: Successfully started service 'sparkDriver' on port 64487.
18/07/10 02:25:48 INFO SparkEnv: Registering MapOutputTracker
18/07/10 02:25:48 INFO SparkEnv: Registering BlockManagerMaster
18/07/10 02:25:48 INFO DiskBlockManager: Created local directory at C:\Users\Mayank\AppData\Local\Temp\blockmgr-28f766e6-9b32-4e63-a0c7-4dacc
18/07/10 02:25:48 INFO MemoryStore: MemoryStore started with capacity 1983.3 MB
18/07/10 02:25:48 INFO SparkEnv: Registering OutputCommitCoordinator
18/07/10 02:25:48 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/07/10 02:25:48 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.70.1:4040
18/07/10 02:25:48 INFO Executor: Starting executor ID driver on host localhost
18/07/10 02:25:48 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 64528.
18/07/10 02:25:48 INFO NettyBlockTransferService: Server created on 192.168.70.1:64528
18/07/10 02:25:48 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.70.1, 64528)
18/07/10 02:25:48 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.70.1:64528 with 1983.3 MB RAM, BlockManagerId(driver, 15
18/07/10 02:25:48 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.70.1, 64528)
Time: 1531169751000 ms
Time: 1531169752000 ms
Time: 1531169753000 ms
Time: 1531169754000 ms
Time: 1531169755000 ms
(#kenedy,1)
(#hescominghome,1)
(#9Jul,1)
(#nufc,1)

```
(#PachaMama,1)
(#happygrandpa,1)
(#weareacmilan,1)
...

-------------------------------------------
Time: 1531169768000 ms
-------------------------------------------
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
(#_1,طيف_الورد_الدهم)
(#PachaMama,1)
(#happygrandpa,1)
(#weareacmilan,1)
...

-------------------------------------------
Time: 1531169769000 ms
-------------------------------------------
(#MasterChef,2)
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
(#_1,طيف_الورد_الدهم)
(#PachaMama,1)
(#happygrandpa,1)
...

-------------------------------------------
Time: 1531169770000 ms
-------------------------------------------
(#MasterChef,2)
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#[redacted],1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
```

Twitter.txt file:



References

- Official documentation of Cloudera Hadoop: https://www.cloudera.com/documentation.html
- Official documentation of Apache Map-Reduce: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- Official documentation of Apache Hive: https://cwiki.apache.org/confluence/display/Hive/LanguageManual
- Official documentation of Apache Sqoop: https://sqoop.apache.org/docs/1.4.6/index.html
- Official documentation of Apache Spark: https://spark.apache.org/docs/2.3.0/