

Week -

Aim: To Implement Mapreduce in java.

Program:

Mapper.java

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WCMapper extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

Reducer.java

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WCReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Driver.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    public static void main(String[] args) throws Exception {
        String in = args[0];
        String out = args[1];

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(WCMapper.class);
        job.setCombinerClass(WCReducer.class);
        job.setReducerClass(WCReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(in));
        FileSystem fs = FileSystem.get(conf); // delete file output when it exists
        if (fs.exists(new Path(out))) {
            fs.delete(new Path(out), true);
        }

        FileOutputFormat.setOutputPath(job, new Path(out));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Command:

hadoop jar wordcount.jar Driver /user/cloudera/words.txt /user/cloudera/output/

Output:

```
Hadoop 1
The    2
This   2
above  1
all    1
alphabets. 1
also   1
article 1
as     1
brown  1
code   1
```

Result: Successfully Implemented Word Count using Mapreduce in java.

Week -

Aim: To Implement Mapreduce in Python.

Program:

Mapper.py

```
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    #clean and split in words
    linechars = [c for c in line.lower() if c.isalpha() or c==' ']
    words = ''.join(linechars).strip().split()

    #emit the key-value pairs
    for word in words:
        print '%s\t%s' % (word, 1)
```

Reducer.py

```
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
```

```

try:
    count = int(count)
except ValueError:
    # count was not a number, so silently
    # ignore/discard this line
    continue

# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
    current_count += count
else:
    if current_word:
        # write result to STDOUT
        print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)

```

Command:

```

$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.3.0.jar \ -
mapper ./mapper.py -reducer ./reducer.py -input wordcount-input -output wordcount-
mapreduce-streaming-python-output

```

```

$HADOOP_HOME/bin/hadoop fs cat wordcount-mapreduce-streaming-python-
output/*

```

Output:

Hadoop	1	
The	2	
This	2	
above	1	
all	1	
alphabets.		1
also	1	
article	1	
as	1	
brown	1	
code	1	

Result: Successfully Implemented Word Count using Mapreduce in Python.

WEEK -

Aim: To Implement Neo4J Graph Database.

Screenshots of execution:

1) Create command

```
neo4j$ CREATE (SV:Student{name: "G.SAI VAMSI", YOB: 2003, POB: "Mangalagiri"})
```

Added 1 label, created 1 node, set 3 properties, completed after 2 ms.

Table

Code

2) Search for some node using match and then creating relations between two nodes (edges)

```
1 MATCH (a:Student), (b:Student_of) WHERE a.name="G.SAI VAMSI" AND b.name = "Vrsec"
2 CREATE (a)-[r: STUDENT_OF]→(b)
3 RETURN a,b
4
```

Graph

Table

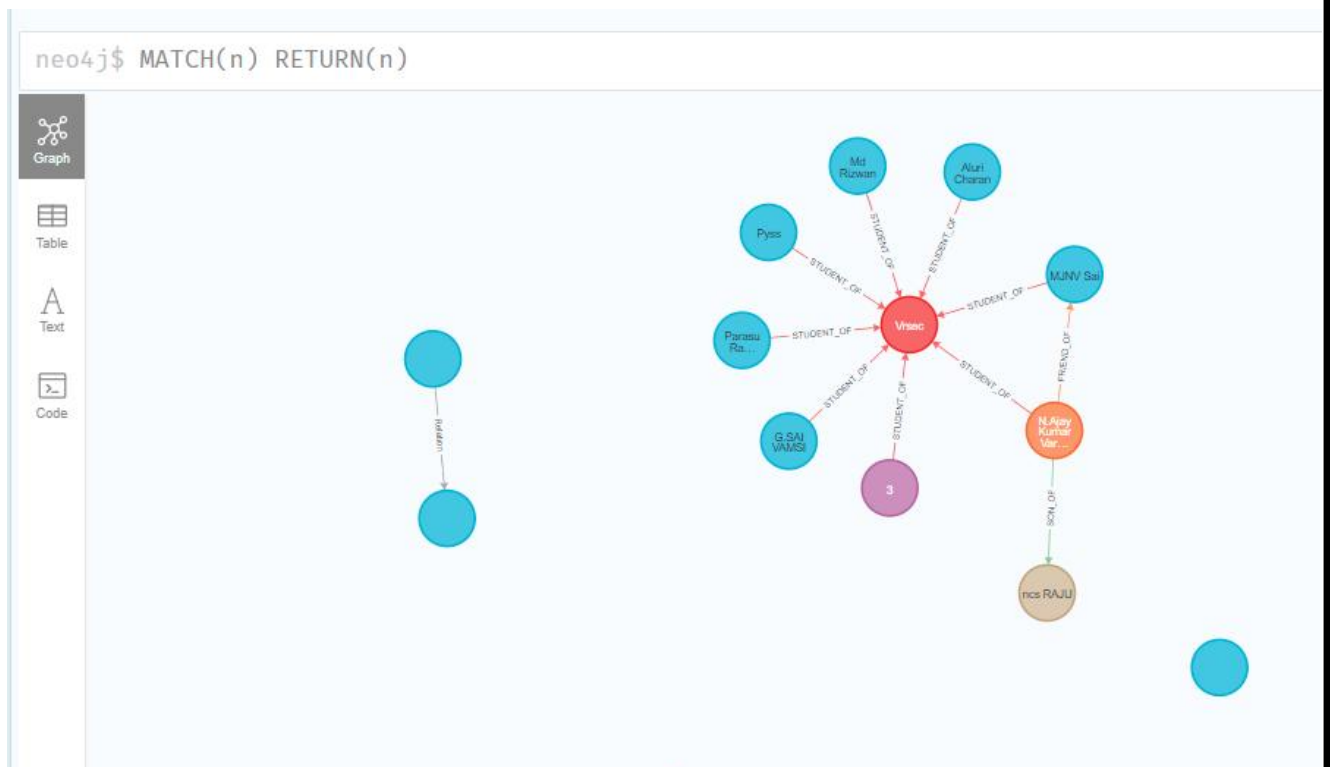
Text

Warn

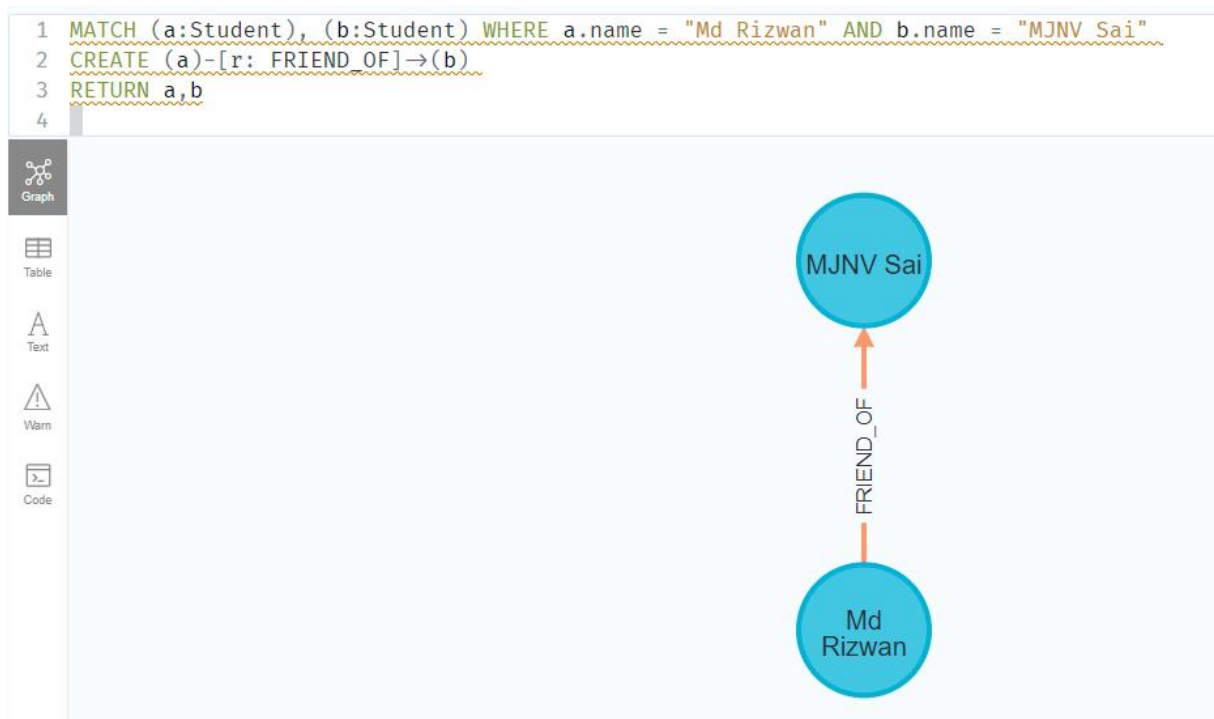
Code

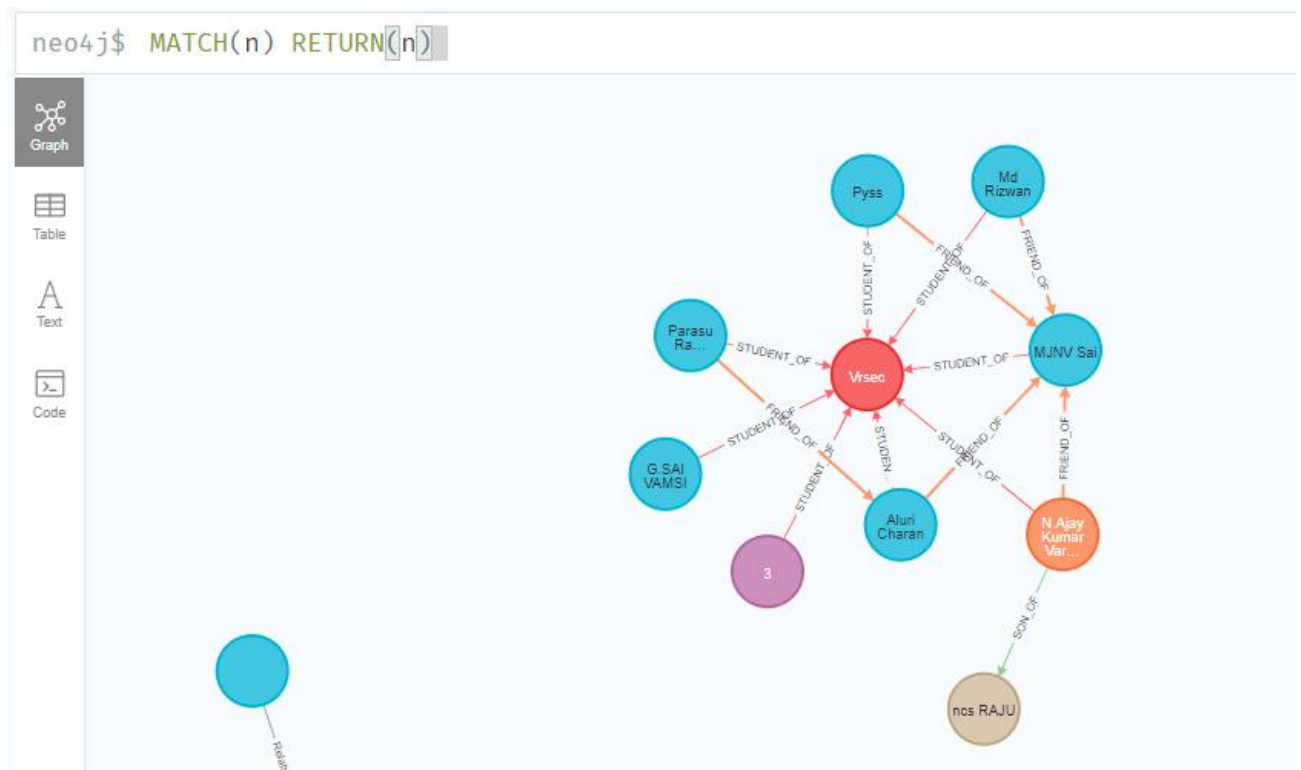
```
graph BT; A((G.SAI VAMSI)) -- STUDENT_OF --> B((Vrsec))
```

3) MATCH(n) RETURN(n) -> Return to display output graph

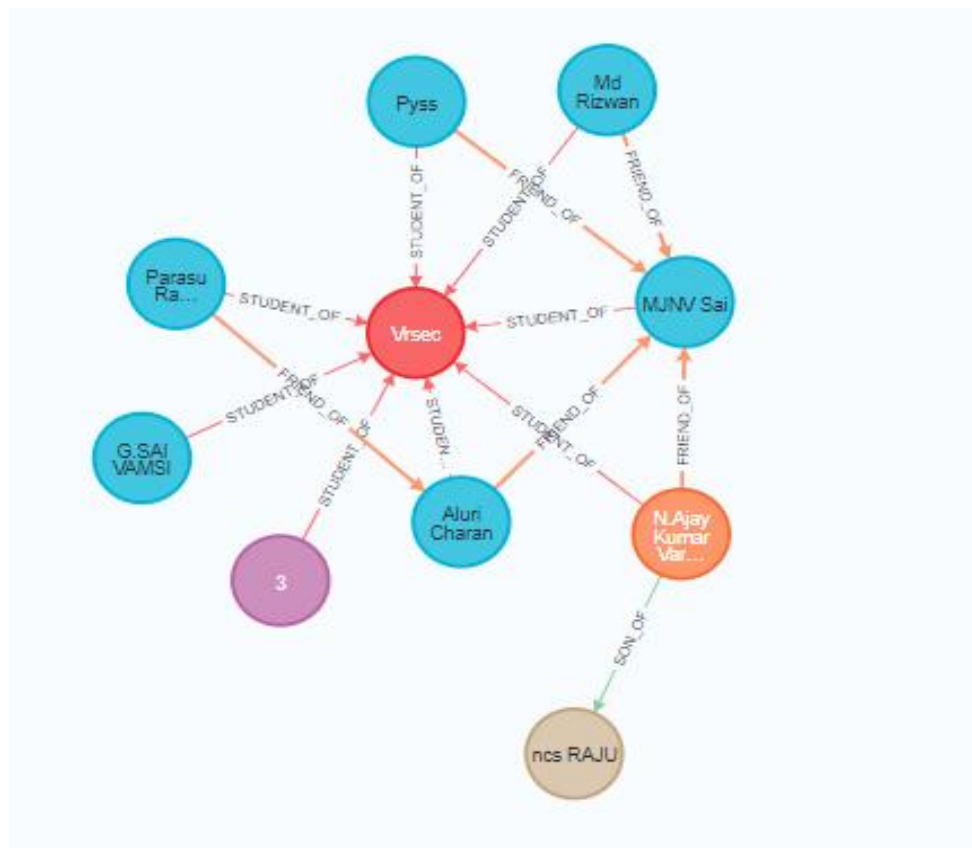


→Creating more edges and establishing Relations between them.





Final Graph Obtained:



Week -

Aim: To Implement Pig Latin.

Procedure:

The Pig Latin statements are used to process the data. It is an operator that accepts a relation as an input and generates another relation as an output.

Pig Latin Simple DataTypes are: int, long, float, double, chararray, bytearray, datetime, Boolean, biginteger, bigdecimal

Pig Latin Complex Data Types: Tuple, Bag, Map

Word count with pig:

The Pig Queries and Results are in below Screenshot

```
grunt> inp= LOAD '/user/cloudera/pigwordcount/wordcount.txt' AS(line:Chararray);
grunt> wordz = FOREACH inp GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;
grunt> wordz = FOREACH inp GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;
grunt> groupedz = GROUP wordz BY word;
grunt> wordcount = FOREACH groupedz GENERATE group,COUNT(wordz);
grunt> dump wordcount
```

Output:

```
2022-12-01 22:31:40,984 [DataStreamer for file /tmp/temp681229446/tmp-551242945/libthrift-0.9.3.jar] WARN org.apache.hadoop.hdfs.DFSClient - Caught exception
java.lang.InterruptedException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1281)
    at java.lang.Thread.join(Thread.java:1355)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
2022-12-01 22:31:41,539 [DataStreamer for file /tmp/temp681229446/tmp1194837738/hive-hcatalog-core-1.11.0-cdh5.13.0.jar] WARN org.apache.hadoop.hdfs.DFSClient - Caught exception
java.lang.InterruptedException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1281)
    at java.lang.Thread.join(Thread.java:1355)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
2022-12-01 22:31:41,579 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.JobControlCompiler - creating jar file Job6699616220070001.jar
2022-12-01 22:31:45,748 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.JobControlCompiler - jar file Job6699616220070001.jar created
2022-12-01 22:31:45,748 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.jar is deprecated. Instead, use mapreduce.job.jar
2022-12-01 22:31:45,807 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.JobControlCompiler - Setting up single store job
2022-12-01 22:31:45,842 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate code.
2022-12-01 22:31:45,842 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distributed cache
2022-12-01 22:31:45,842 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Setting key [pig.schematuple.classes] with classes to deserialize []
2022-12-01 22:31:46,051 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - 1 map-reduce job(s) waiting for submission.
2022-12-01 22:31:46,058 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker.http.address is deprecated. Instead, use mapreduce.jobtracker.http.address
2022-12-01 22:31:46,058 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 22:31:46,073 [JobControl] INFO org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2022-12-01 22:31:46,121 [JobControl] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 22:31:46,943 [JobControl] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-12-01 22:31:46,943 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 1
2022-12-01 22:31:46,908 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths (combined) to process : 1
2022-12-01 22:31:47,079 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:1
2022-12-01 22:31:47,486 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_1669961622007_0001
2022-12-01 22:31:48,390 [JobControl] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application application_1669961622007_0001
2022-12-01 22:31:48,454 [JobControl] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: http://quickstart.cloudera:8088/proxy/application_1669961622007_0001/
2022-12-01 22:31:48,454 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - HadoopJobId: job_1669961622007_0001
2022-12-01 22:31:48,454 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - Processing aliases grouped,lines,wordcount,words
2022-12-01 22:31:48,454 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - detailed locations: W: lines[1,8],words[-1,-1],wordcount[4,12],grouped[3,10] C: wordcount[4,12],grouped[3,10] R: wordscount[4,12]
2022-12-01 22:31:48,454 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - More information at: http://localhost:58038/jobdetails.jsp?jobid=job_1669961622007_0001
2022-12-01 22:31:48,541 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - 0% complete
2022-12-01 22:32:14,111 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - 50% complete
2022-12-01 22:32:29,208 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreducelayer.MapReduceLauncher - 100% complete
2022-12-01 22:32:29,212 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script Statistics:

HadoopVersion PigVersion UserId StartedAt FinishedAt Features
2.6.0-cdh5.13.0 0.12.0-cdh5.13.0 cloudera 2022-12-01 22:31:40 2022-12-01 22:32:29 GROUP_BY
```

```

Input(s):
Successfully read 12 records (529 bytes) from: "/user/cloudera/pigwordcount/wordcount.txt"

Output(s):
Successfully stored 13 records (149 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp845550026/tmp-1025923344"

Counters:
Total records written : 13
Total bytes written : 149
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1669961666269_0004

2022-12-01 22:51:43,304 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-12-01 22:51:43,305 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 22:51:43,305 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 22:51:43,306 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-12-01 22:51:43,318 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-12-01 22:51:43,319 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(sai,2)
(ajay,3)
(mjnv,2)
(star,1)
(aluri,2)
(annam,1)
(chand,1)
(jitin,2)
(kumar,2)
(varma,1)
(charan,2)
(rizwan,2)
(mohammad,2)
grunt> █

```

Result: Successfully Implemented Word Count using Pig Latin.

Week -

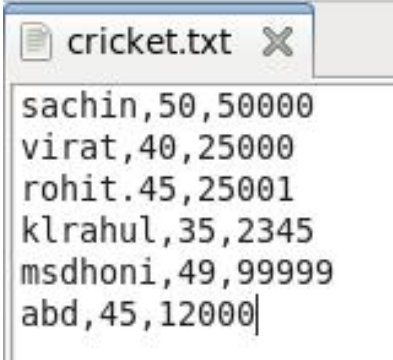
Aim: To Implement Hive.

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.

Below are Screenshots of Queries and Execution.

- 1) Create a new database and then a new table using Hive DDL commands.
- 2) Loading data with same datatypes from a text or a csv file using below queries.

```
hive> create table players(playername string,player_age int,player_runs int) row format delimited fields terminated by
OK
Time taken: 0.525 seconds
hive> show tables;
OK
players
Time taken: 0.055 seconds, Fetched: 1 row(s)
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/cricket.txt' INTO TABLE players;
Loading data to table cricket.players
Table cricket.players stats: [numFiles=1, totalSize=92]
OK
Time taken: 1.067 seconds
hive> select * from players;
OK
sachin 50      50000
virat  40      25000
rohit.45 25001    NULL
klrahul 35      2345
msdhoni 49      99999
abd     45      12000
Time taken: 0.423 seconds, Fetched: 6 row(s)
```



The screenshot shows a text editor window titled 'cricket.txt'. The content of the file is a CSV representation of cricket player data, with each line representing a player's name, age, and runs. The data is as follows:

Player Name	Age	Runs
sachin	50	50000
virat	40	25000
rohit.45	25001	NULL
klrahul	35	2345
msdhoni	49	99999
abd	45	12000

Fig: Text Present inside Text File

Implementing Word count using Hive:

Load data to hive and then write query present in below screenshot.

```
hive> show tables;
OK
demostr
players
Time taken: 0.021 seconds, Fetched: 2 row(s)
hive> LOAD DATA LOCAL INPATH "/home/cloudera/Desktop/wordcount.txt" overwrite into table demostr;
Loading data to table cricket.demostr
Table cricket.demostr stats: [numFiles=1, numRows=0, totalSize=138, rawDataSize=0]
OK
Time taken: 0.649 seconds
hive> select * from demostr;
OK
ajay kumar varma
ajay kumar
mjnv sai
mjnv
charan aluri
aluri
charan
rizwan mohammad
mohammad rizwan
sai ajay
star jitin
annam jitin chand
Time taken: 0.112 seconds, Fetched: 12 row(s)
hive> create table word_count as select word,count(1) as count from (select explode(split(line,'\s')) as word from demostr) w group by word order by word;
Query ID = cloudera_20221201234646_051b07c7-7103-477b-978f-a3fea0fb5338
Time taken: 0.095 seconds, Fetched: 14 row(s)
```

Output:

```
hive> select * from word_count;
OK
          2
ai          1
ai ajay 1
ajay kumar          1
ajay kumar varma          1
aluri 1
annam jitin chand          1
charan 1
charan aluri 1
mjnv 1
mjnv 1
mohammad rizwan 1
rizwan mohammad 1
tar jitin 1
Time taken: 0.095 seconds, Fetched: 14 row(s)
```

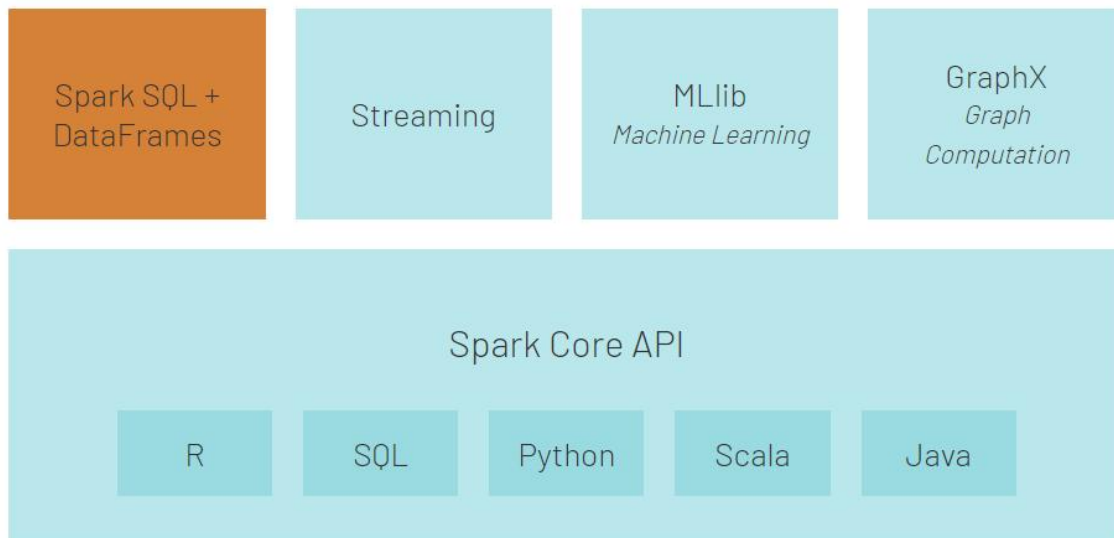
Result: Successfully Implemented Hive.

Week -

Aim: To Implement Apache Spark in DataBricks.

Apache Spark is a lightning-fast unified analytics engine for big data and machine learning.

Apache Spark Eco System



Word Count using Spark in Data Bricks

```
line count in text file Scala
File Edit View Run Help Last edit was 6 minutes ago Give feedback

Cmd 1
1 %python
2 display(dbutils.fs.ls("/FileStore/tables"))

> (3) Spark Jobs
Table +
path name size
1 dbfs:/FileStore/tables/text.txt text.txt 64
Showing 1 row | 4.52 seconds runtime
Command took 4.52 seconds -- by 208w1a1268@vrsec.ac.in at 03/12/2022, 15:16:09 on Cluster for sparkml

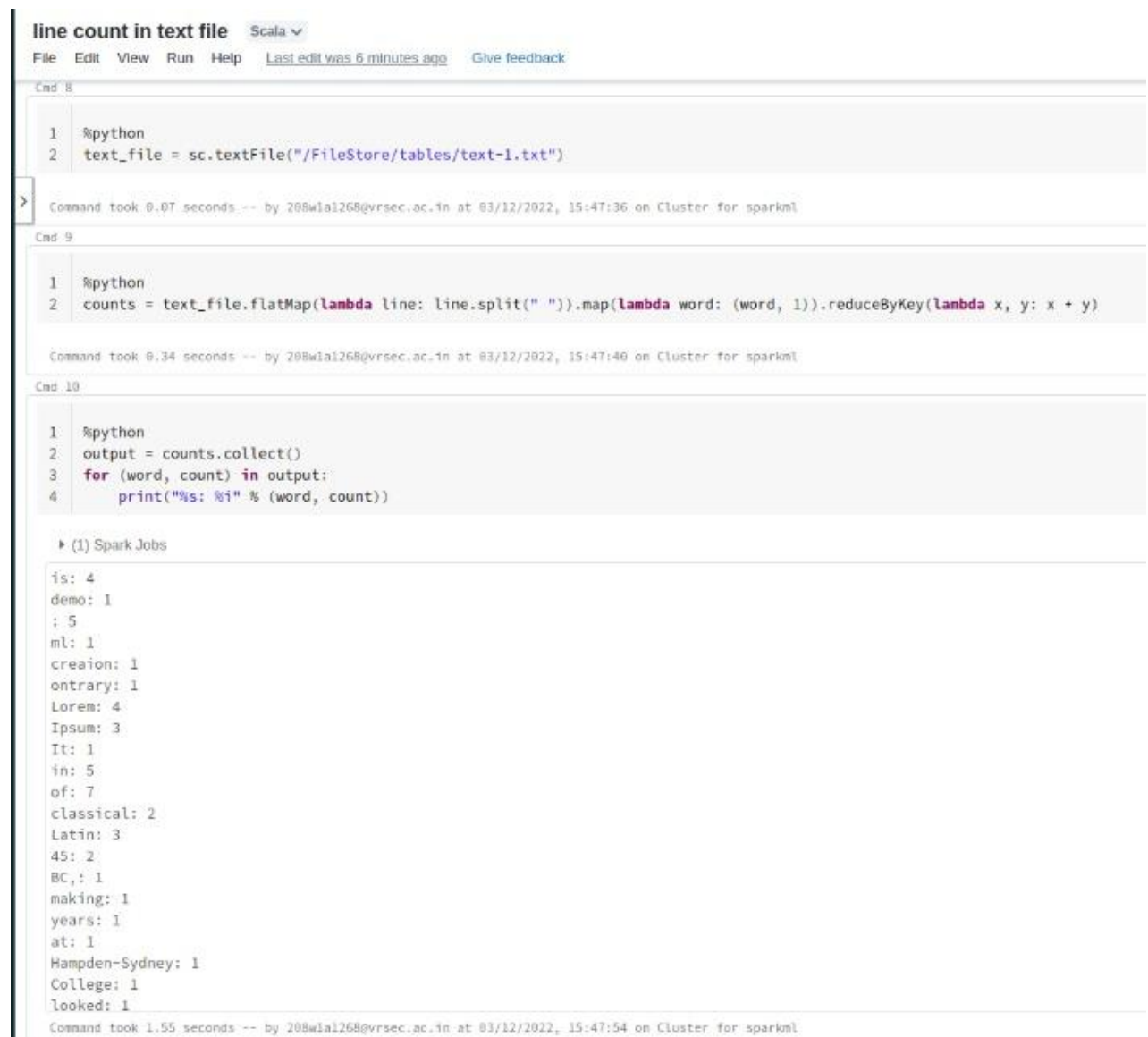
Cmd 2
1 %python
2 textFile = spark.read.text("/FileStore/tables/text.txt")
textFile: pyspark.sql.dataframe.DataFrame = [value:string]
Command took 1.58 seconds -- by 208w1a1268@vrsec.ac.in at 03/12/2022, 15:21:40 on Cluster for sparkml

Cmd 3
1 %python
2 textFile.count()
(2) Spark Jobs
Out[9]: 3
Command took 2.97 seconds -- by 208w1a1268@vrsec.ac.in at 03/12/2022, 15:22:06 on Cluster for sparkml

Cmd 4
1 %python
2 textFile.first()
(1) Spark Jobs
Out[10]: Row(value='This is the demo data for spark ')
Command took 0.49 seconds -- by 208w1a1268@vrsec.ac.in at 03/12/2022, 15:23:20 on Cluster for sparkml

Cmd 5
```

Output:



The screenshot shows a Databricks notebook titled "line count in text file" with a Scala language dropdown. It contains three code cells and their corresponding outputs.

Cell 8: A code cell with two lines of Python code: `%python` and `text_file = sc.textFile("/FileStore/tables/text-1.txt")`. The output shows the command took 0.07 seconds.

Cell 9: A code cell with two lines of Python code: `%python` and `counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda x, y: x + y)`. The output shows the command took 0.34 seconds.

Cell 10: A code cell with four lines of Python code: `%python`, `output = counts.collect()`, `for (word, count) in output:`, and `print("%s: %i" % (word, count))`. The output shows the command took 1.55 seconds and displays the results of the word count.

Output of Cell 10: A list of words and their counts, preceded by a Spark Jobs summary: `(1) Spark Jobs`. The output is: `is: 4`, `demo: 1`, `: 5`, `ml: 1`, `creaion: 1`, `ontrary: 1`, `lorem: 4`, `Ipsum: 3`, `It: 1`, `in: 5`, `of: 7`, `classical: 2`, `Latin: 3`, `45: 2`, `BC,: 1`, `making: 1`, `years: 1`, `at: 1`, `Hampden-Sydney: 1`, `College: 1`, `looked: 1`.

Fig: Results of Word count using Spark

Result: Successfully Implemented Spark in DataBricks.

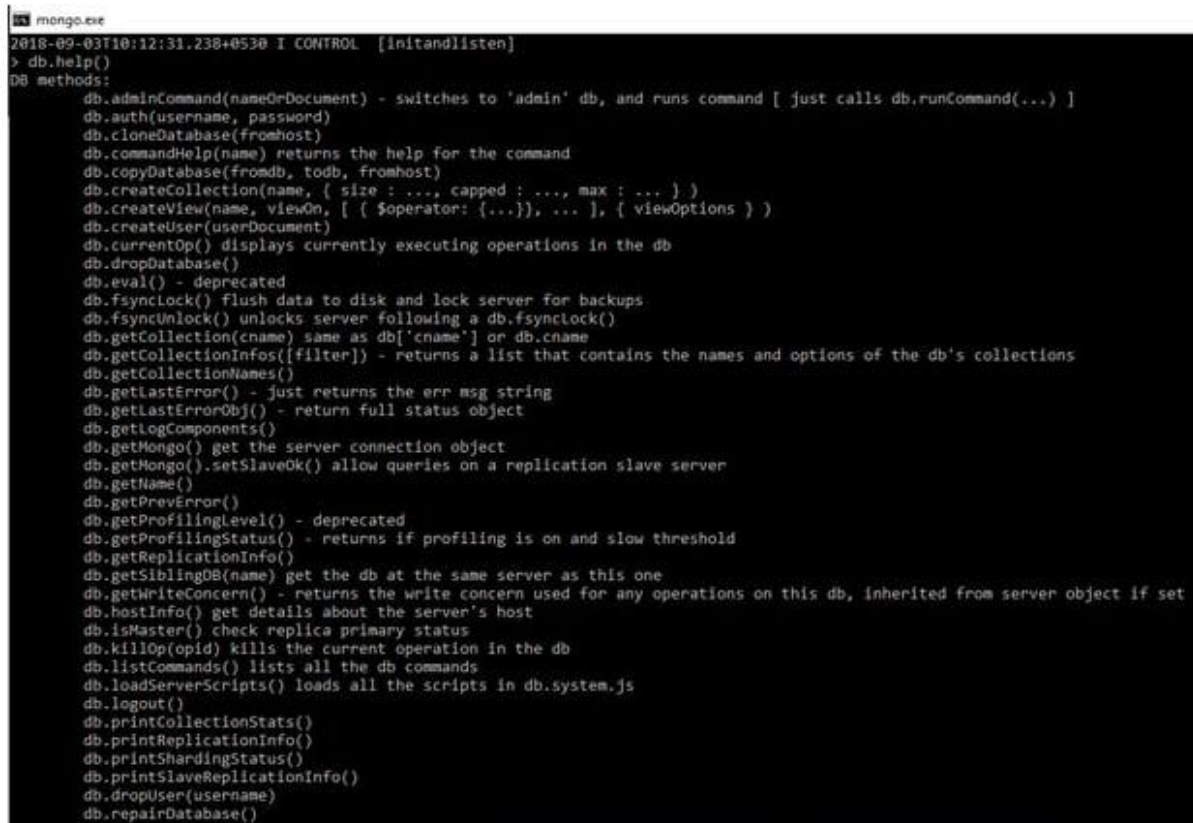
Week -

Aim: To Implement Mongo Db.

Commands Below:

To get a list of commands, type `db.help()` in MongoDB client. This will give you a list of commands as shown in the following screenshot.

`db.help`



```
mongo.exe
2018-09-03T10:12:31.238+0530 I CONTROL [initandlisten]
> db.help()
DB methods:
  db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runCommand(...) ]
  db.auth(username, password)
  db.cloneDatabase(fromhost)
  db.commandHelp(name) returns the help for the command
  db.copyDatabase(fromdb, todb, fromhost)
  db.createCollection(name, { size : ..., capped : ..., max : ... } )
  db.createView(name, viewOn, [ { $operator: {...}}, ... ], { viewOptions } )
  db.createUser(userDocument)
  db.currentOp() displays currently executing operations in the db
  db.dropDatabase()
  db.eval() - deprecated
  db.fsyncLock() flush data to disk and lock server for backups
  db.fsyncUnlock() unlocks server following a db.fsyncLock()
  db.getCollection(cname) same as db['cname'] or db.cname
  db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections
  db.getCollectionNames()
  db.getLastErrorMessage() - just returns the err msg string
  db.getLastErrorMessageObj() - return full status object
  db.getLogComponents()
  db.getMongo() get the server connection object
  db.getMongo().setSlaveOk() allow queries on a replication slave server
  db.getName()
  db.getPrevError()
  db.getProfilingLevel() - deprecated
  db.getProfilingStatus() - returns if profiling is on and slow threshold
  db.getReplicationInfo()
  db.getSiblingDB(name) get the db at the same server as this one
  db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set
  db.hostInfo() get details about the server's host
  db.isMaster() check replica primary status
  db.killOp(opid) kills the current operation in the db
  db.listCommands() lists all the db commands
  db.loadServerScripts() loads all the scripts in db.system.js
  db.logout()
  db.printCollectionStats()
  db.printReplicationInfo()
  db.printShardingStatus()
  db.printSlaveReplicationInfo()
  db.dropUser(username)
  db.repairDatabase()
```

Show All Databases

Use below command to get list of all databases.

`show dbs`


```
sai> sai> show dbs
admin      40.00 KiB
config    108.00 KiB
local      40.00 KiB
sai        88.00 KiB
sai>
```

Create new database

To create a new database execute the following command.

`use DATABASE_NAME`

```
CA mongo.exe
> use myTestDB
switched to db myTestDB
>
```

To know your current working/selected database execute the following command

`db`

```
> db
myTestDB
>
```

Drop database

To drop the database execute following command, this will drop the selected database

`db.dropDatabase()`

```
> db.dropDatabase()
{ "dropped" : "myTestDB", "ok" : 1 }
>
```

Insert document in collection

`>db.COLLECTION_NAME.insert(document)`

```
sai> db.sai.insertMany([ {name : "vijay" , age : "20" } , { name : "venu" , age : "35" } ] );
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63493373d1d5d10758f8df08"),
    '1': ObjectId("63493373d1d5d10758f8df09")
  }
}
```

DISPLAY THE RECORDS :

```
sai> db.sai.find().pretty() ;
[
  {
    _id: ObjectId("63493279d1d5d10758f8df07"),
    name: 'vijay',
    age: '20'
  },
  {
    _id: ObjectId("63493373d1d5d10758f8df08"),
    name: 'vijay',
    age: '20'
  },
  {
    _id: ObjectId("63493373d1d5d10758f8df09"),
    name: 'vijay',
    age: '35'
  }
]
```

```
sai> db.createCollection('abcd') ;
{ ok: 1 }
sai> db.sai.insert( {name : "vijay" , age : "20" } , { name : "venu" , age : "35" } ) ;
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("63493279d1d5d10758f8df07") }
}
```

UPDATE THE RECORDS :

```
sai> ssai> ssai> sai> db.sai.update ( {"name" : "venu" }, { $set: {"name" : "vijay" } } ); sai> db.sai.update ( {"name" : "venu" }, { $set: {"name" : "vijay" } } );
```

acknowledged: true,
insertedId: null,
matchedCount: 0,
modifiedCount: 0,
upsertedCount: 0

```
sai> db.sai.insertMany([ {name : "vijay" , age : "20" } , { name : "venu" , age : "35" } ] );
```

{
 acknowledged: true,
 insertedIds: {
 '0': ObjectId("63493373d1d5d10758f8df08"),
 '1': ObjectId("63493373d1d5d10758f8df09")
 }
}

Result: Successfully Implemented MongoDB commands.