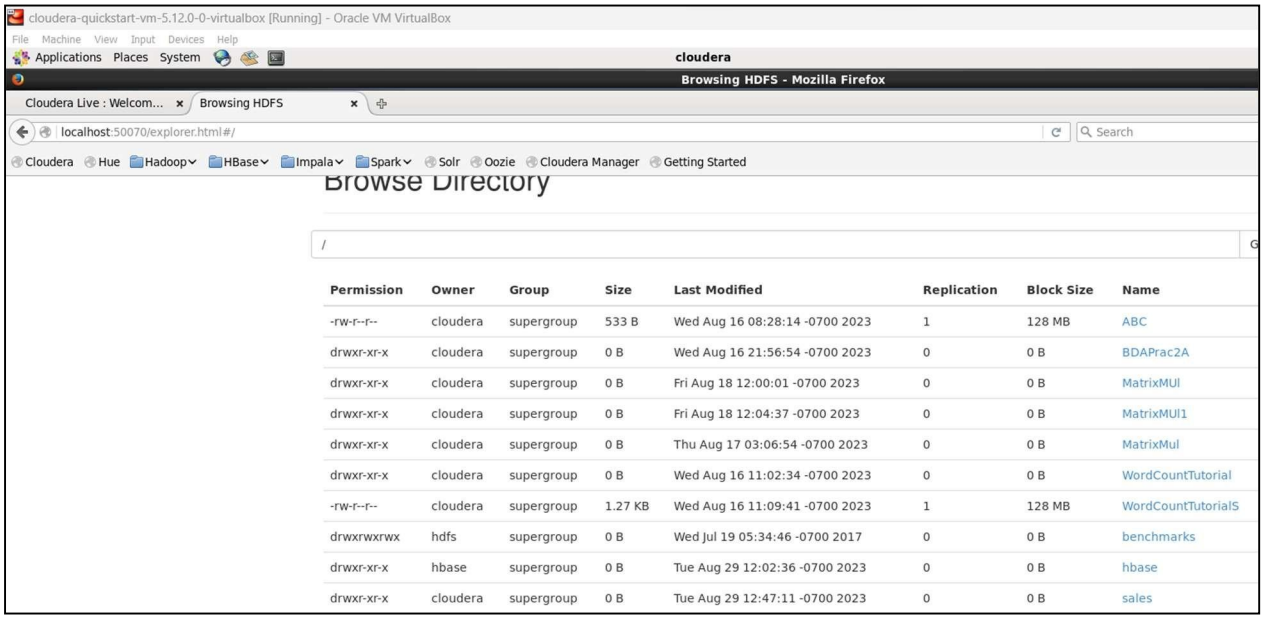
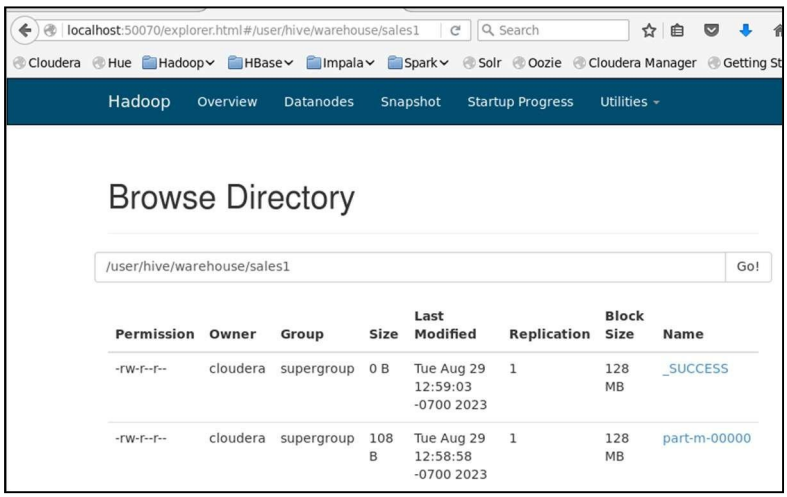
Importing tables from RDMS to HDFS using Sqoop:

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost/sales --username=root --
password="cloudera" --table=sales1 --target-dir=/sales/sales -incremental append --check-column
 month_number --fields-terminated-by='\t';
```

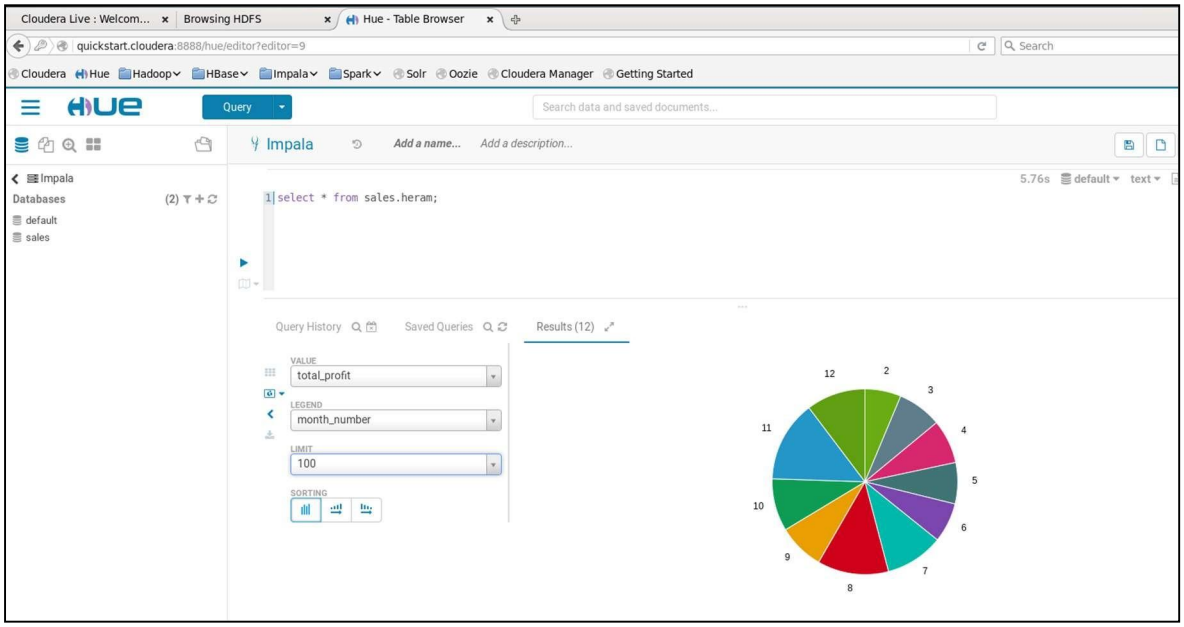


Importing Table From HDFS to HIVE:

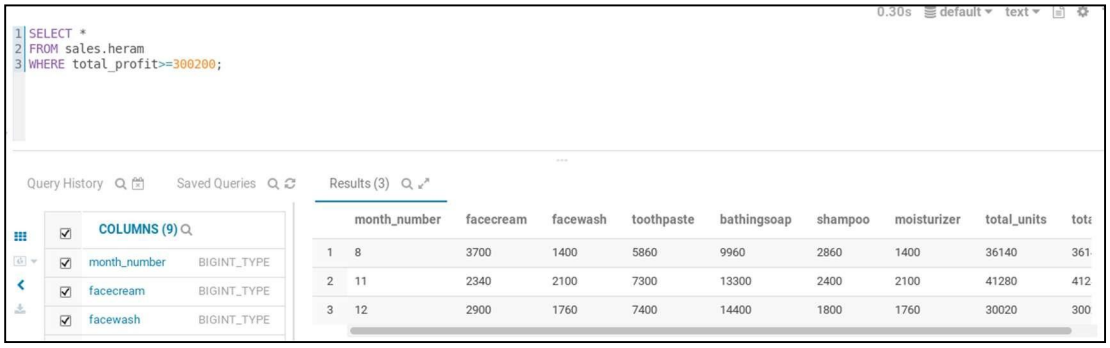
```
[cloudera@quickstart ~]$ sqoop import-all-tables --connect jdbc:mysql://localhost/sales --username
root --password "cloudera" --warehouse-dir /user/hive/warehouse
```

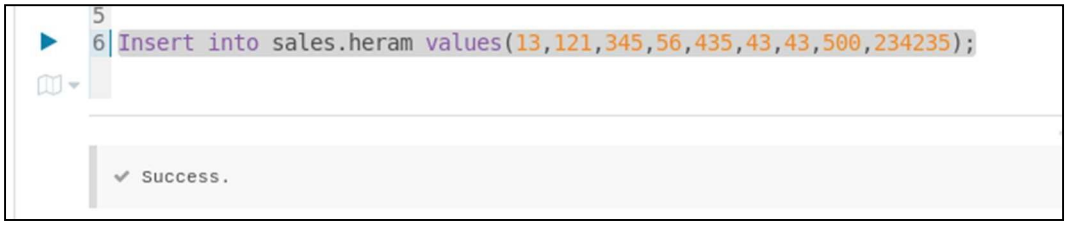


Going to Hue Editor, Importing table, Writing Query And Doing Visualization.



Running Some Queries:





Output:

```
[cloudera@quickstart ~]$ mysql -uroot -pcloudera
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> CREATE DATABASE sales;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> use sales;
Database changed
```

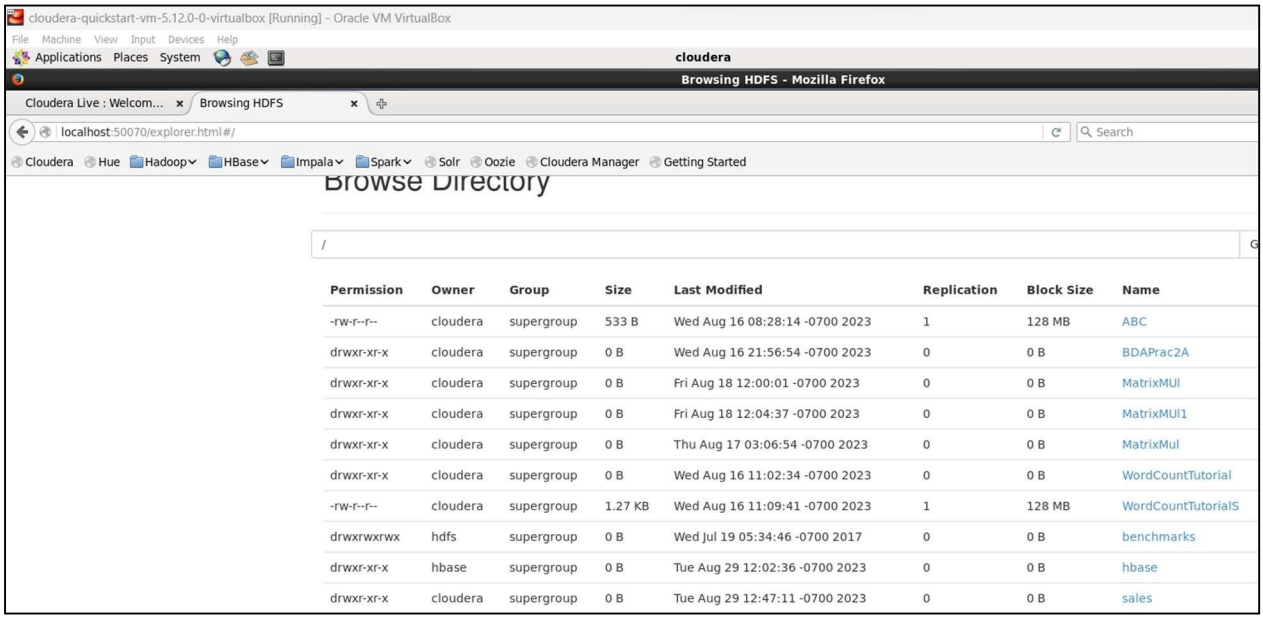
```
mysql> LOAD DATA Local Infile '/home/cloudera/Desktop/Heramb/Heram.csv 'into table sales1 Fields Terminated By ',' Li
nes Terminated By '\n';
Query OK, 13 rows affected, 9 warnings (0.02 sec)
Records: 13  Deleted: 0  Skipped: 0  Warnings: 0
```

```
mysql> select * from sales1;
+--------------+-----------+----------+------------+-------------+---------+-------------+-------------+-------------
-+
| month_number | facecream | facewash | toothpaste | bathingsoap | shampoo | moisturizer | total_units | total_profit
 |
+--------------+-----------+----------+------------+-------------+---------+-------------+-------------+-------------
-+
|            0 |         0 |        0 |          0 |           0 |       0 |           0 |           0 |            0
 |
|            1 |      2500 |     1500 |       5200 |        9200 |    1200 |        1500 |       21100 |       211000
 |
|            2 |      2630 |     1200 |       5100 |        6100 |    2100 |        1200 |       18330 |       183300
 |
|            3 |      2140 |     1340 |       4550 |        9550 |    3550 |        1340 |       22470 |       224700
 |
|            4 |      3400 |     1130 |       5870 |        8870 |    1870 |        1130 |       22270 |       222700
 |
|            5 |      3600 |     1740 |       4560 |        7760 |    1560 |        1740 |       20960 |       209600
 |
```

```
mysql> show tables
    -> ;
+------------------+
| Tables_in_sales |
+------------------+
| sales1           |
+------------------+
```

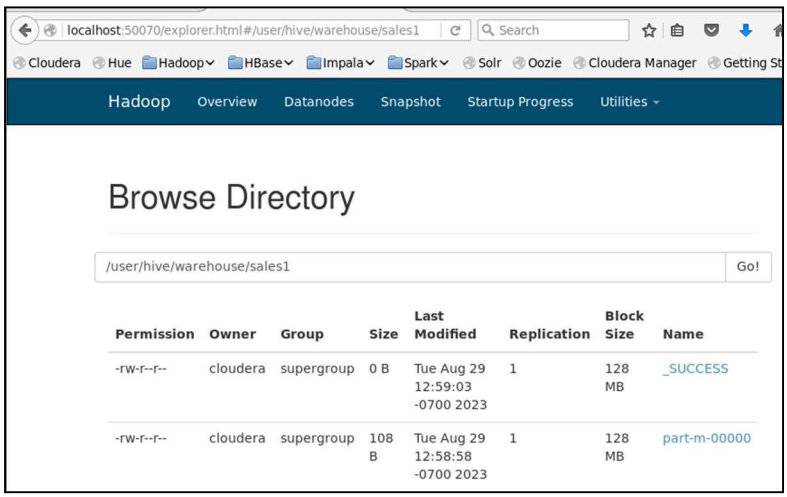
Importing tables from RDMS to HDFS using Sqoop:

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost/sales --username=root --
password="cloudera" --table=sales1 --target-dir=/sales/sales -incremental append --check-column
 month_number --fields-terminated-by='\t';
```

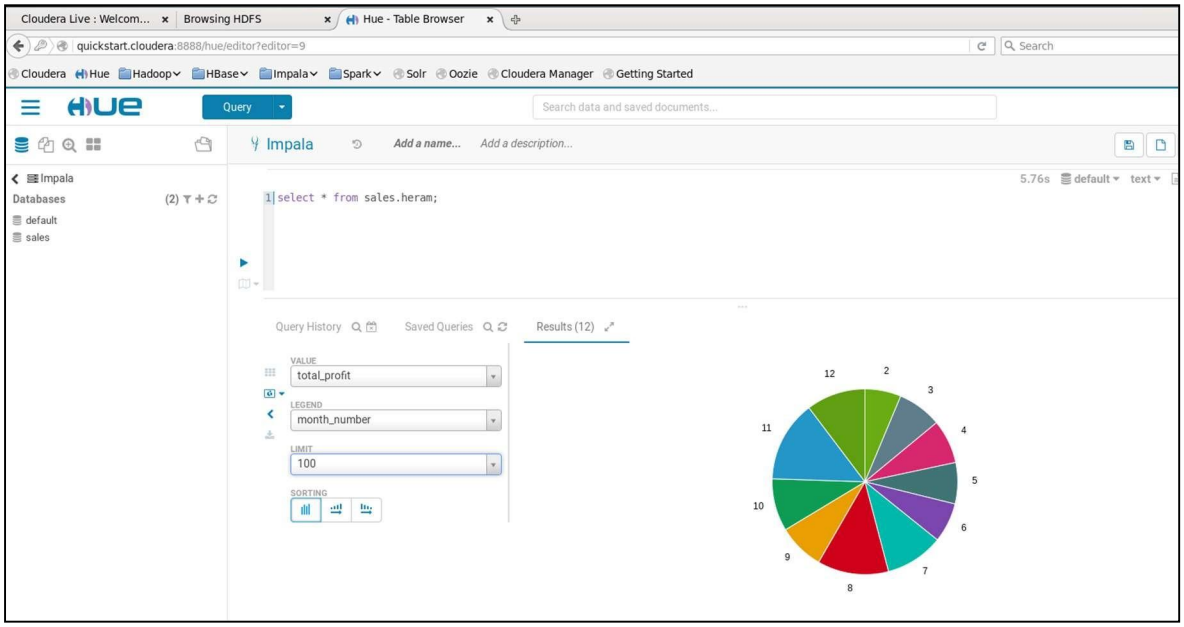


Importing Table From HDFS to HIVE:

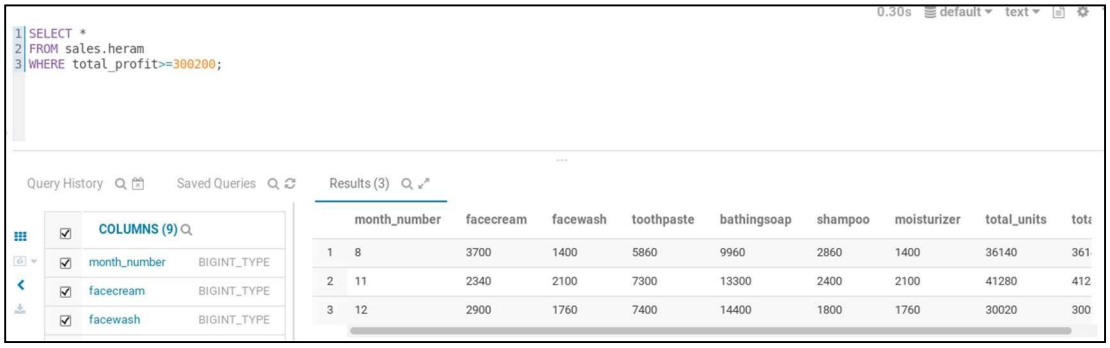
```
[cloudera@quickstart ~]$ sqoop import-all-tables --connect jdbc:mysql://localhost/sales --username
root --password "cloudera" --warehouse-dir /user/hive/warehouse
```

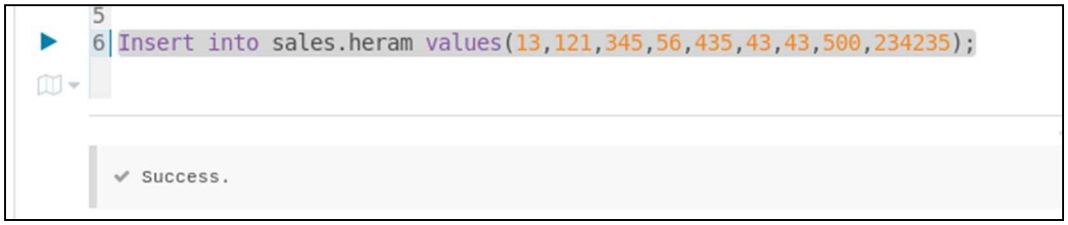


Going to Hue Editor, Importing table, Writing Query And Doing Visualization.



Running Some Queries:





Program:

First Type 'pyspark' in the terminal then type the below commands.

```
>>> sc.appName
u'PySparkShell'
>>>from pyspark import SparkConf, SparkContext
>>> sc
<pyspark.context.SparkContext object at 0x2918c50>
>>> rdd1=sc.textFile("file:/home/cloudera/RT/data1.txt")
>>> rdd2=rdd1.flatMap(lambda line:line.split())
>>> rdd3=rdd2.filter(lambda word:word.startswith('h'))
>>> rdd4=rdd3.map(lambda word:(word,1))
>>> rdd4.collect
```

Output:

```
>>> sc.appName
u'PySparkShell'
>>> from pyspark import SparkConf, SparkContext
>>> sc
<pyspark.context.SparkContext object at 0x1285c50>
>>> rdd1=sc.textFile("file:/home/cloudera/Desktop/BDAPrac2A/Heramb.txt")
>>> rdd2=rdd1.flatMap(lambda line:line.split())
>>> rdd3=rdd2.filter(lambda word:word.startswith('H'))
>>> rdd4=rdd3.map(lambda word:(word,1))
>>> rdd4.collect()
[(u'Hi', 1), (u"Heramb's", 1), (u'Himanshu', 1), (u'Help', 1), (u'He', 1), (u'Help', 1), (u'He', 1), (u'Help', 1)]
```

```
>>> sc.appName
u'PySparkShell'
>>> from pyspark import SparkConf, SparkContext
>>> sc
<pyspark.context.SparkContext object at 0x1285c50>
>>> rdd1=sc.textFile("file:/home/cloudera/Desktop/BDAPrac2A/Heramb.txt")
>>> rdd2=rdd1.flatMap(lambda line:line.split())
>>> rdd3=rdd2.filter(lambda word:word.startswith('A'))
>>> rdd4=rdd3.map(lambda word:(word,1))
>>> rdd4.collect()
[(u'Anjali', 1), (u'Arnav', 1)]
```

Program + Output RDD Programs

A. Selection

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df = sqlContext.read.json("/user/cloudera/iris.json")
df.show()
df.select("species").show()
df.select(df['petalLength'], df['species'] + 1).show()
```

```
+-----------+-------------+
|petalLength|(species + 1)|
+-----------+-------------+
|       null|         null|
|        1.4|         null|
|        1.4|         null|
|        1.3|         null|
|        1.5|         null|
|        1.4|         null|
|        1.7|         null|
|        1.4|         null|
|        1.5|         null|
```

B. Projection

```
>>> from pyspark import SparkContext
>>> c=sc.parallelize([["name","gender","age"],["A","Male","20"],["B","Female","21"],["C","Male","23"],["D","Female","
25"]])
>>> c.collect()
[['name', 'gender', 'age'], ['A', 'Male', '20'], ['B', 'Female', '21'], ['C', 'Male', '23'], ['D', 'Female', '25']]
>>> test=c.map(lambda x: x[0])
>>> print "projection ->%s" %(test.collect())
projection ->['name', 'A', 'B', 'C', 'D']
>>> test=c.map(lambda x:x[1])
>>> print "projection ->%s" %(test.collect())
projection ->['gender', 'Male', 'Female', 'Male', 'Female']
```

C. Union

```
>>> sqlContext=SQLContext(sc)
>>> valuesB=[('abc',1),('pqr',2),('mno',7),('xyz',9)]
>>> TableB=sqlContext.createDataFrame(valuesB,['name','customerid'])
>>> valuesC=[('abc',1),('pqr',2),('mno',7),('efg',10),('hik',12)]
>>> TableC=sqlContext.createDataFrame(valuesC,['name','customerid'])
>>> result=TableB.unionAll(TableC)
>>> result.show()
+----+----------+
|name|customerid|
+----+----------+
| abc|         1|
| pqr|         2|
| mno|         7|
| xyz|         9|
| abc|         1|
| pqr|         2|
| mno|         7|
| efg|        10|
| hik|        12|
+----+----------+
```

D. Aggregate and Grouping

Sum:

```
>>> data=[[1,2],[2,1],[4,3],[4,5],[5,4],[1,4],[1,1]]
>>> list1=sc.parallelize(data)
>>> list1.collect()
[[1, 2], [2, 1], [4, 3], [4, 5], [5, 4], [1, 4], [1, 1]]
>>>
>>>
>>> mapped_list=list1.map(lambda x: (x[0],x[1]))
>>> summation=mapped_list.reduceByKey(lambda x,y: x+y)
>>> summation.collect()
[(1, 7), (2, 1), (4, 8), (5, 4)]
```

Average:

```
>>> input1=sqlContext.createDataFrame([(1,2),(2,6),(1,8),(2,4),(3,1),(3,1),(3,1)],["col1","col2"])
>>> input1.groupBy("col1").agg({"col2":"avg"}).show()
+----+---------+
|col1|avg(col2)|
+----+---------+
|   1|      5.0|
|   2|      5.0|
|   3|      1.0|
+----+---------+
```

```
>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>> df = sqlContext.read.json("/user/cloudera/iris.json")
>>> df.groupBy("species").agg({"petalLength": "avg"}).show()
+----------+------------------+
|   species|  avg(petalLength)|
+----------+------------------+
|versicolor|              4.26|
|    setosa|1.4620000000000002|
| virginica|             5.552|
|      null|              null|
+----------+------------------+
```

Count:

```
>>> mapped_count = df.map(lambda x : (x[-1],1))
>>> count = mapped_count.reduceByKey(lambda x,y : x+y)
>>> count.collect()
[(None, 2), (u'setosa', 50), (u'versicolor', 50), (u'virginica', 50)]
```

Max & Min element

```
>>> max_element=mapped_list.reduceByKey(lambda x,y:max(x,y))
>>> max_element.collect()
[(1, 4), (2, 1), (4, 5), (5, 4)]
>>>
>>> min_element=mapped_list.reduceByKey(lambda x,y:min(x,y))
>>> min_element.collect()
[(1, 1), (2, 1), (4, 3), (5, 4)]
```

E. Join

```
>>> valueA=[('Pasta',1),('Pizza',2),('Spaghetti',3),('Rice',4)]
>>> rdd1=sc.parallelize(valueA)
>>> TableA=sqlContext.createDataFrame(rdd1,['name','id'])
>>>
>>>
>>> valueB=[('White',1),('Red',2),('Pasta',3),('Spaghetti',4)]
>>> rdd2=sc.parallelize(valueB)
>>> TableB=sqlContext.createDataFrame(rdd2,['name','id'])
>>>
>>> TableA.show()
+---------+---+
|     name| id|
+---------+---+
|    Pasta|  1|
|    Pizza|  2|
|Spaghetti|  3|
|     Rice|  4|
+---------+---+

>>>
>>> TableB.show()
+---------+---+
|     name| id|
+---------+---+
|    White|  1|
|      Red|  2|
|    Pasta|  3|
|Spaghetti|  4|
+---------+---+

>>> ta=TableA.alias('ta')
>>> tb=TableB.alias('tb')
```

```
>>> inner_join=ta.join(tb,ta.name==tb.name)
>>> inner_join.show()
+---------+---+---------+---+
|     name| id|     name| id|
+---------+---+---------+---+
|Spaghetti|  3|Spaghetti|  4|
|    Pasta|  1|    Pasta|  3|
+---------+---+---------+---+

>>>
>>> left=ta.join(tb,ta.name==tb.name,how='left')
>>> left.show()
+---------+---+---------+----+
|     name| id|     name|  id|
+---------+---+---------+----+
|     Rice|  4|     null|null|
|Spaghetti|  3|Spaghetti|   4|
|    Pasta|  1|    Pasta|   3|
|    Pizza|  2|     null|null|
+---------+---+---------+----+

>>> right=ta.join(tb,ta.name==tb.name,how='right')
>>> right.show()
+---------+----+---------+---+
|     name|  id|     name| id|
+---------+----+---------+---+
|Spaghetti|   3|Spaghetti|  4|
|     null|null|    White|  1|
|    Pasta|   1|    Pasta|  3|
|     null|null|      Red|  2|
+---------+----+---------+---+
```

F. Intersection

```
>>> input1=sc.textFile("file:/home/cloudera/Desktop/BDAPrac2A/input1.txt")
>>> mapinput1=input1.flatMap(lambda x:x.split(","))
>>> mapinput1.collect()
[u'Hello', u' this', u' is', u' Heramb', u' Practical']
>>>
>>> input2=sc.textFile("file:/home/cloudera/Desktop/BDAPrac2A/input2.txt")
>>> mapinput2=input2.flatMap(lambda x:x.split(","))
>>> mapinput2.collect()
[u'Hello', u' this', u' is', u' Heramb', u' Assignment']
>>>
>>>
>>> input3=mapinput1+mapinput2
>>> input3.collect()
[u'Hello', u' this', u' is', u' Heramb', u' Practical', u'Hello', u' this', u' i
s', u' Heramb', u' Assignment']
>>>
>>>
>>> finalintersection=input3.map(lambda word:(word,1))
>>> finalintersection.collect()
[(u'Hello', 1), (u' this', 1), (u' is', 1), (u' Heramb', 1), (u' Practical', 1),
 (u'Hello', 1), (u' this', 1), (u' is', 1), (u' Heramb', 1), (u' Assignment', 1)
]
>>> joiningfinalintersection=finalintersection.reduceByKey(lambda x,y:(x+y))
>>> joiningfinalintersection.collect()
[(u' Heramb', 2), (u' Assignment', 1), (u' this', 2), (u' Practical', 1), (u' is
', 2), (u'Hello', 2)]
>>>
>>>
>>> finalans=joiningfinalintersection.filter(lambda x:x[1]>1)
>>> finalans.collect()
[(u' Heramb', 2), (u' this', 2), (u' is', 2), (u'Hello', 2)]
```

Program:

A. To implement the word Count

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

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

 public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  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);
    }
  }
 }
```

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

  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);
  }
 }

 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Output

```
[cloudera@quickstart BDAPrac2A]$ hadoop dfs -cat /BDAPrac2A/Output/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

After   1
Completed       1
Efforts 1
Hardwork.       1
Heramb's        1
It      1
Lot     1
Of      1
Practical.      1
This    1
Was     1
and     1
is      1
```

B. Matrix-Vector multiplication

Programs:

```
public class MatrixVectorMultiplication {
    public static void main(String[] args) {
        // Define the matrix and vector
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[] vector = {2, 3, 4};

        // Check if matrix and vector dimensions are compatible
        int matrixRows = matrix.length;
        int matrixCols = matrix[0].length;
        int vectorSize = vector.length;

        if (matrixCols != vectorSize) {
            System.out.println("Matrix and vector dimensions are not compatible for
multiplication.");
            return;
        }

        // Perform matrix-vector multiplication
        int[] result = new int[matrixRows];
        for (int i = 0; i < matrixRows; i++) {
```

```
            for (int j = 0; j < matrixCols; j++) {
                result[i] += matrix[i][j] * vector[j];
            }
        }

        // Display the result
        System.out.println("Result of matrix-vector multiplication:");
        for (int i = 0; i < matrixRows; i++) {
            System.out.println(result[i]);
        }
    }
}
```

Output:

```
Result of matrix-vector multiplication:
20
47
74
```

Results and Discussions:

MapReduce is a parallel processing model for large scale data**:**

Word Count:

Result: Efficiently count words occurrences in texts.
Discussions: Map phase splits text, emits . Reduce phase aggregate counts. Scales well for basic tasks.

Matrix-Vector Multiplication: