Important Instructions:

- Please read the document thoroughly before you code.
- Please do not change the Business Requirements.

Coverage:

1. HDFS
2. Map Reduce
3. HIVE
4. PIG Script

For all the hands-on, please consider the following,

UNIX User name: training

Home Directory: /home/training

HDFS

Requirement 1:

You are supposed to create a directory name "samples" in the home directory and create two files named "emp" and "dept" as follows,

Emp:

EMPNO,ENAME,JOB,SAL,DEPTNO
7369,SMITH,CLERK,800,20
7499,ALLEN,SALESMAN,1600,30
7521,WARD,SALESMAN,1250,30
7566,JONES,MANAGER,2975,20
7654,MARTIN,SALESMAN,1400,30
7698,BLAKE,MANAGER,2850,30
7782,CLARK,MANAGER,2450,10
7788,SCOTT,ANALYST,3000,20
7839,KING,PRESIDENT,5000,10
7844,TURNER,SALESMAN,1500,30

7876,ADAMS,CLERK,1100,20
7900,JAMES,CLERK,950,30
7902,FORD,ANALYST,3000,20
7934,MILLER,CLERK,1300,10

Dept :
DEPTNO,DNAME,LOC
10,ACCOUNTING,NEWYORK
20,RESEARCH,DALLAS
30,SALES,CHICAGO
40,OPERATIONS,BOSTON

```
hduser@hadoop:~$ mkdir samples
hduser@hadoop:~$ ls
customer.avsc   Desktop    hello.txt                      nc.scala~         samples
customer.java   Documents  kafka-logs                     Pictures          scala_out
customer:name}  Downloads  mongodb_inst_steps_4.0_14.04~  Public            spark_out
dataset         error~     Music                          QueryResult.java  Templates
hduser@hadoop:~$ cd samples
hduser@hadoop:~/samples$ nano emp.txt
hduser@hadoop:~/samples$ nano dept.txt
hduser@hadoop:~/samples$ ls
dept.txt  emp.txt
```

Now create a directory named "data" in HDFS. Copy the "samples" directory into HDFS. Now write commands

1. to see the content of the samples directory in HDFS

```
hduser@hadoop:~$ hdfs dfs -mkdir /data
hduser@hadoop:~$ hdfs dfs -copyFromLocal /home/hduser/samples /data/
hduser@hadoop:~$ hdfs dfs -ls  /data/samples
Found 2 items
-rw-r--r--   1 hduser supergroup         97 2021-05-28 15:28 /data/samples/dept.txt
-rw-r--r--   1 hduser supergroup        402 2021-05-28 15:28 /data/samples/emp.txt
hduser@hadoop:~$
```

2. to see the first 5 employees from the "emp" file located in HDFS

```
hduser@hadoop:~$ hdfs dfs -cat /data/samples/emp.txt  | head -5
EMPNO,ENAME,JOB,SAL,DEPTNO
7369,SMITH,CLERK,800,20
7499,ALLEN,SALESMAN,1600,30
7521,WARD,SALESMAN,1250,30
7566,JONES,MANAGER,2975,20
hduser@hadoop:~$
```

3. to remove the "dept "file located in HDFS

```
hduser@hadoop:~$ hdfs dfs -rm /data/samples/dept.txt
21/05/28 15:32:34 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minute
, Emptier interval = 0 minutes.
Deleted /data/samples/dept.txt
hduser@hadoop:~$ hdfs dfs -ls /data/samples
Found 1 items
-rw-r--r--   1 hduser supergroup        402 2021-05-28 15:28 /data/samples/emp.txt
hduser@hadoop:~$
```

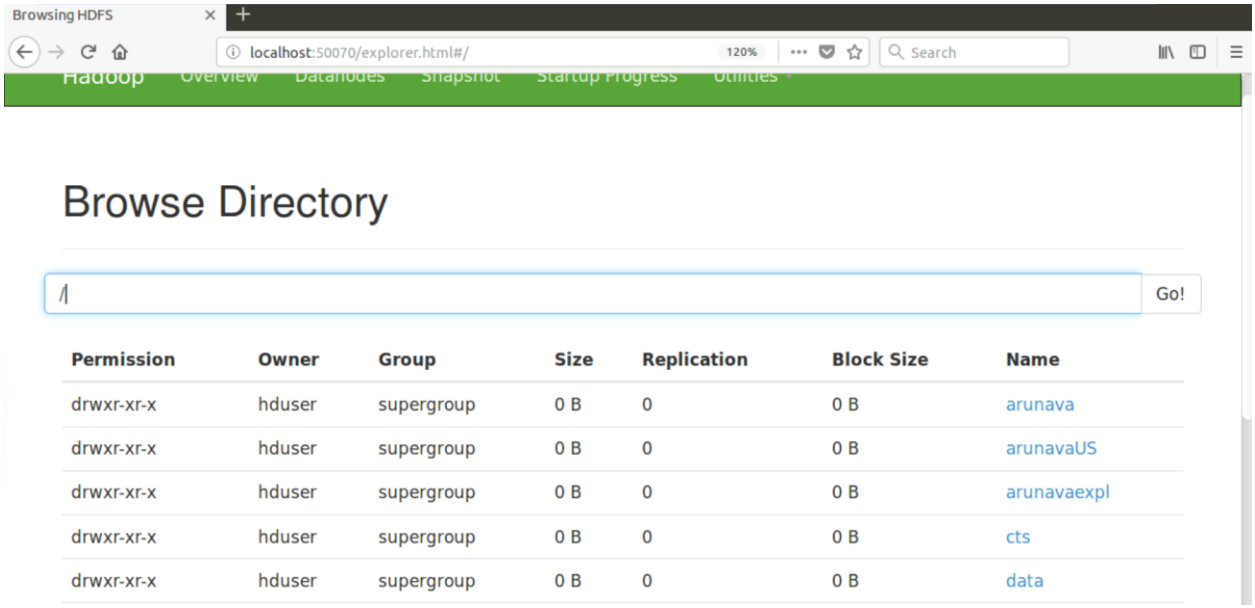
4. to copy the "emp" file from HDFS to the home directory in Unix

```
hduser@hadoop:~$ hdfs dfs -copyToLocal   /data/samples/emp.txt  /home/hduser/
hduser@hadoop:~$ ls
customer.avsc   Documents  kafka-logs                  Public          Templates
customer.java   Downloads  mongodb_inst_steps_4.0_14.04~  QueryResult.java  u.data
customer:name}  emp.txt    Music                       samples         Videos
dataset         error~     nc.scala~                   scala_out       workspace_scala
Desktop         hello.txt  Pictures                    spark_out       zookeeper
hduser@hadoop:~$
```

5. Open the browser, type in the following URL

   http://localhost:50070

   And browse the file system



Map Reduce

Requirement 2:

We asked few Questions and the response of the Batch Representative i s

Q1 Name the Creators in the class

Ans : Sasi, Sreejith, Vijay

Q2 Name few who solves all activities

Ans : Dharan, Varsha, Gowtham

Q3 Who are quick learners in the class

Ans: Sasi, Sreejith, Vijay

Q4: Who takes more effort to solve all activities?

Ans: Pavan, Rajesh, Surya, Aasik

Q5. Who come by cab to office?

Ans: Varsha, Aasik, Surya

Q6: Who all have 100% Attendance in the class?

Ans: Sreejith, Varsha, Pavan

Problem Statement: The above Q & A should be used as the input and Need to count the number of occurrence of each name

Stage 1: Mapper

Task 1: Splitting the input to get the Key-Values

Key	Values
Creators :	Sasi, Sreejith, Vijay
Developers	Dharan, Varsha, Gowtham
Quick Learners :	Sasi, Sreejith, Vijay
Warriors :	Pavan, Rajesh, Surya, Aasik
Travellers :	Varsha, Aasik, Surya
MrPunctual :	Sreejith, Varsha, Pavan

Task 2: Mapping - Each key would be mapped to a Value and count would be 1

Creators :	Sasi	1
Creators :	Sreejith	1
Creators :	Vijay	1
Developers :	Dharan	1
Developers :	Varsha	1
Developers :	Gowtham	1
Quick Learners :	Sasi	1
Quick Learners :	Sreejith	1
Quick Learners :	Vijay	1
Warriors :	Pavan	1
Warriors :	Rajesh	1
Warriors :	Surya	1
Warriors :	Aasik	1
Travellers :	Varsha	1
Travellers :	Aasik	1
Travellers :	Surya	1
MrPunctual :	Sreejith	1
MrPunctual :	Varsha	1

MrPunctual : Pavan 1

Stage 2: Reducer

Task 1: Shuffling so that values occur as a group

Creators: Sasi 1
Quick Learners: Sasi 1

Creators: Sreejith 1
Quick Learners: Sreejith 1
MrPunctual: Sreejith 1

Creators: Vijay 1
Quick Learners: Vijay 1

Developers: Dharan 1

Developers: Varsha 1
Travellers: Varsha 1
MrPunctual: Varsha 1

Developers : Gowtham 1

Warriors : Pavan 1
MrPunctual : Pavan 1

Warriors : Rajesh 1

Warriors : Surya 1
Travellers : Surya 1

Warriors : Aasik 1
Travellers : Aasik 1

Now we can start with the following code to perform the Stage 1 - Task 1, which is to construct a Map with Key-Value pairs.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.*;

public class FileRead {
public static void main ( String[] args ) throws FileNotFoundException
{
        Scanner scanner = new Scanner(new FileReader("D:/neuron.txt"));
        Map<String, ArrayList> myMap = new HashMap<String, ArrayList>();
        String line;

         while (scanner.hasNextLine())
        {
                line = scanner.nextLine();
                if (!line.isEmpty())
                {
                        String[] columns = line.split("=");
                        String k = columns[0];
                        String[] v = columns[1].split(" ");
                        System.out.println("Key : " + k);
                        ArrayList al = new ArrayList<String>();
                        for ( int i=0; i<v.length; i++)
                        {
                                System.out.println("Value : " + v[i]);
                                al.add(v[i]);
                        }
                        myMap.put(k, al);
                }
        }
        System.out.println ( myMap );
}
}
```

Complete the Code to implement Map-Reduce by writing a Driver class and configure your Mapper and Reducer. Specify the input and output directories. Please make sure you are using a unique non-existent output directory.

```
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.conf.Configuration;

public class CreateTable {

  public static void main(String[] args) throws IOException {

    // Instantiating configuration class
    Configuration con = HBaseConfiguration.create();

    // Instantiating HbaseAdmin class
    HBaseAdmin admin = new HBaseAdmin(con);

    // Instantiating table descriptor class
    HTableDescriptor tableDescriptor = new
    HTableDescriptor(TableName.valueOf("emp"));

    // Adding column families to table descriptor
    tableDescriptor.addFamily(new HColumnDescriptor("personal"));
    tableDescriptor.addFamily(new HColumnDescriptor("professional"));

    // Execute the table through admin
    admin.createTable(tableDescriptor);
    System.out.println(" Table created ");
```

```
  }
}
```

HIVE

Requirement 3:

Create a Database named MyDB in HIVE and create a table named "students" with the following columns

regno	NUMERIC	4
sname	VARCHAR	20
mark1	NUMERIC	3
mark2	NUMERIC	3
mark3	NUMERIC	3

We have a file named STUDENT.TXT in the samples directory of the Home Directory (/home/training/samples) with following data.

1000,'RAJ',56,76,91
1001,'AMIT',96,86,91
1002,'DINESH',82,81,84
1003,'MAHESH',77,78,79
1004,'DAVID',86,86,81
1005,'JONES',67,76,71
1006,'ANKIT',88,83,98
1007,'ANAND',89,87,94

```
hive> use MyDB;
OK
Time taken: 2.461 seconds
hive> show tables;
OK
Time taken: 0.457 seconds
hive> create table students (regno int,sname VARCHAR(20),mark1 int,mark2 int,mar
k3 int)row format delimited fields terminated by ',';
OK
Time taken: 0.649 seconds
hive> load data local inpath '/home/hduser/samples/STUDENT.txt' into table stude
nts;
Loading data to table mydb.students
Table mydb.students stats: [numFiles=1, totalSize=177]
OK
Time taken: 1.734 seconds
```

You are supposed to write a command to load the data from student.txt which is in the local file system.

1. Write a command to view the contents of the table students on the console.

Ans: **select * from students;**

```
ive> select * from students;
K
000     'RAJ'    56      76      91
001     'AMIT'   96      86      91
002     'DINESH'         82      81      84
003     'MAHESH'         77      78      79
004     'DAVID'  86      86      81
005     'JONES'  67      76      71
006     'ANKIT'  88      83      98
007     'ANAND'  89      87      94
ULL     NULL     NULL    NULL    NULL
ULL     NULL     NULL    NULL    NULL
ime taken: 0.64 seconds, Fetched: 10 row(s)
ive> exit;
```

2. Write a command to view the schema of the Table

Ans: **describe students;**

```
hive> describe students;
OK
regno                   int
sname                   varchar(20)
mark1                   int
mark2                   int
mark3                   int
Time taken: 0.333 seconds, Fetched: 5 row(s)
hive>
```

3. Write a command to view the size of the table and other information.

   Ans: **describe formatted students;**

```
hive> describe formatted students;
OK
# col_name              data_type               comment

regno                   int
sname                   varchar(20)
mark1                   int
mark2                   int
mark3                   int

# Detailed Table Information
Database:               mydb
Owner:                  hduser
CreateTime:             Sat May 29 13:23:59 IST 2021
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://localhost:50000/user/hive/warehouse/mydb.db/students
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   true
        numFiles                1
        totalSize               177
        transient_lastDdlTime   1622274874

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
```

Requirement 4:

Create a HIVE table named "EMP" with the following columns

EMPNO	NUMERIC	4
ENAME	VARCHAR	20
JOB	VARCHAR	20
SAL	NUMERIC	6
DEPTNO	NUMERIC	2

We have a "emp" file available in HDFS (We have created this file in Hands-on 1).

Write a command to load the data from "emp" file in HDFS and load into EMP table in Hive.

Ans: **create table EMP(EMPNO int,ENAME VARCHAR (20),JOB VARCHAR (20),SAL int,DEPTNO int) row format delimited fields terminated by ',' ;**

```
hive> create table EMP(EMPNO int,ENAME VARCHAR (20),JOB VARCHAR (20),SAL int,DEPTNO int) row format delimited
 fields terminated by ',' ;
OK
Time taken: 0.499 seconds
hive> describe EMP;
OK
empno                   int
ename                   varchar(20)
job                     varchar(20)
sal                     int
deptno                  int
Time taken: 0.141 seconds, Fetched: 5 row(s)
hive> load data inpath '/data/samples/emp.txt' into table EMP;
Loading data to table mydb.emp
Table mydb.emp stats: [numFiles=1, totalSize=402]
OK
Time taken: 0.82 seconds
```

Ans: **load data inpath '/data/samples/emp.txt' into table EMP;**

```
hive> select * from EMP;
OK
NULL    ENAME   JOB     NULL    NULL
7369    SMITH   CLERK   800     20
7499    ALLEN   SALESMAN        1600    30
7521    WARD    SALESMAN        1250    30
7566    JONES   MANAGER 2975    20
7654    MARTIN  SALESMAN        1400    30
7698    BLAKE   MANAGER 2850    30
7782    CLARK   MANAGER 2450    10
7788    SCOTT   ANALYST 3000    20
7839    KING    PRESIDENT       5000    10
7844    TURNER  SALESMAN        1500    30
7876    ADAMS   CLERK   1100    20
7900    JAMES   CLERK   950     30
7902    FORD    ANALYST 3000    20
7934    MILLER  CLERK   1300    10
NULL    NULL    NULL    NULL    NULL
Time taken: 0.524 seconds, Fetched: 16 row(s)
hive>
```

Requirement 5:

We have already created and loaded a table named "emp" in the previous Activity. Now create a Java JDBC program to read the data from the emp table and display all the employees.

The template of the Java JDBC program is given below,

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class readEmpHive
{       private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
        public static void main(String[] args) throws SQLException
        {
                // Register driver and create driver instance
                Class.forName(driverName);
                // get connection
                Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/MyDB",
"", "");
                // create statement
                Statement stmt = con.createStatement();
                // execute statement
                Resultset res = stmt.executeQuery( << Place your Query >> );

                System.out.println(" Empno \t EName \t Job \t Salary \t Deptno ");

                << Place your code block to display all employee details >>

                con.close();
        }
}
```

Save the program in a file named readEmpHive.java. Use the following commands to compile and execute this program.

javac readEmpHive.java
java readEmpHive

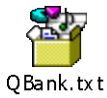
Ans:

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class readEmpHive
{       private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
        public static void main(String[] args) throws SQLException
        {
                // Register driver and create driver instance
                Class.forName(driverName);
                // get connection
                Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/MyDB",
"","");
                // create statement
                Statement stmt = con.createStatement();
                // execute statement
                Resultset res = stmt.executeQuery("SELECT * FROM emp;");

                System.out.println(" Empno \t EName \t Job \t Salary \t Deptno ");
         while (res.next())
          {
           System.out.println(res.getInt(1) + " " + res.getString(2) + " " + res.getString(3) + " " +
res.getInt(4) + " " + res.getInt(5));
          }
                con.close();
        }
```

}

Requirement 6:

We have a file named "QBank" located in the samples directory of the Home directory and following is the content of the file.



Create a Hive table as follows,

create table QB (words string);

Write a command to load the contents of the QBank file into the table name QB.

Ans**: LOAD DATA LOCAL INPATH '/home/hduser/samples/QBank.txt' OVERWRITE INTO TABLE QB;**

```
hive> select * from QB;
OK
Name
Name
Who
Who
Who
Copycats
Sleepers
Time taken: 0.421 seconds, Fetched: 7 row(s)
```

Write a Hive command to count the frequency of each word from the QB and load them into a new Hive table named word_count which has the following two columns

word	VARCHAR	20
wcount	NUMERIC	2

Ans: **CREATE TABLE word_count(word varchar(20),wcount int) row format delimited fields terminated by ',';**

insert overwrite table word_count

SELECT word, count(1) AS count FROM
(SELECT explode(split(words, ' ')) AS word FROM QB) w
GROUP BY word
ORDER BY word;

```
hive> insert overwrite table word_count
    > SELECT word, count(1) AS count FROM
    > (SELECT explode(split(words, ' ')) AS word FROM QB) w
    > GROUP BY word
    > ORDER BY word;
Query ID = hduser_20210529134806_03e17c2d-56b8-4304-9e7f-1506614e32cf
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1622276186590_0001, Tracking URL = http://hadoop:8088/proxy/a
pplication_1622276186590_0001/
Kill Command = /usr/local/hadoop-2.6.0/bin/hadoop job  -kill job_1622276186590_0
001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
```

```
hive> select * from word_count;
OK
        8
Copycats        1
Name    2
Sleepers        1
Who     3
activities      2
all     2
assessment      1
by      1
cab     1
class   3
come    1
dont    1
few     1
in      4
listen  1
office  1
solve   1
solves  1
stupids 1
the     5
to      2
try     1
who     1
Time taken: 0.207 seconds, Fetched: 24 row(s)
```

Requirement 7:

We have a file named "product" located in the samples directory of the Home directory and following is the content of the file.

Write a Hive command to create an External table named "product".

Ans: **create external table product(col1 int,col2 varchar(20),col3 int,col4 int) row format delimited fields terminated by ',' location '/home/hduser/product_details';**

```
hive> create external table product(col1 int,col2 varchar(20),col3 int,col4 int)
 row format delimited fields terminated by ','
    > location '/home/hduser/product_details';
OK
Time taken: 0.618 seconds
hive> describe formatted product;
OK
# col_name              data_type               comment

col1                    int
col2                    varchar(20)
col3                    int
col4                    int

# Detailed Table Information
Database:               mydb
Owner:                  hduser
CreateTime:             Sat May 29 14:19:29 IST 2021
LastAccessTime:         UNKNOWN
Protect Mode:           None
```

```
# Detailed Table Information
Database:               mydb
Owner:                  hduser
CreateTime:             Sat May 29 14:19:29 IST 2021
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://localhost:50000/home/hduser/product_details
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        transient_lastDdlTime   1622278169

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputForm
at
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
```

Ans:

load data local inpath '/home/hduser/samples/product.txt' into table product;

select * from product;

```
Time taken: 1.854 seconds
hive> select * from product;
OK
1       Shirt   10      1000
2       Shoe    20      2000
3       Trouser 10      2500
4       Umbrella        30      250
5       Pen     30      50
6       Pencil  30      10
7       Cabbage 50      60
8       Fish    40      100
9       Carrot  50      50
10      Beans   50      65
NULL    NULL    NULL    NULL
Time taken: 0.572 seconds, Fetched: 11 row(s)
hive>
```

Requirement 8:

We have the following 2 Activity log files which contains the browsing history of Wikipedia under the samples directory of the home directory


browse_history 1


browse_history 2

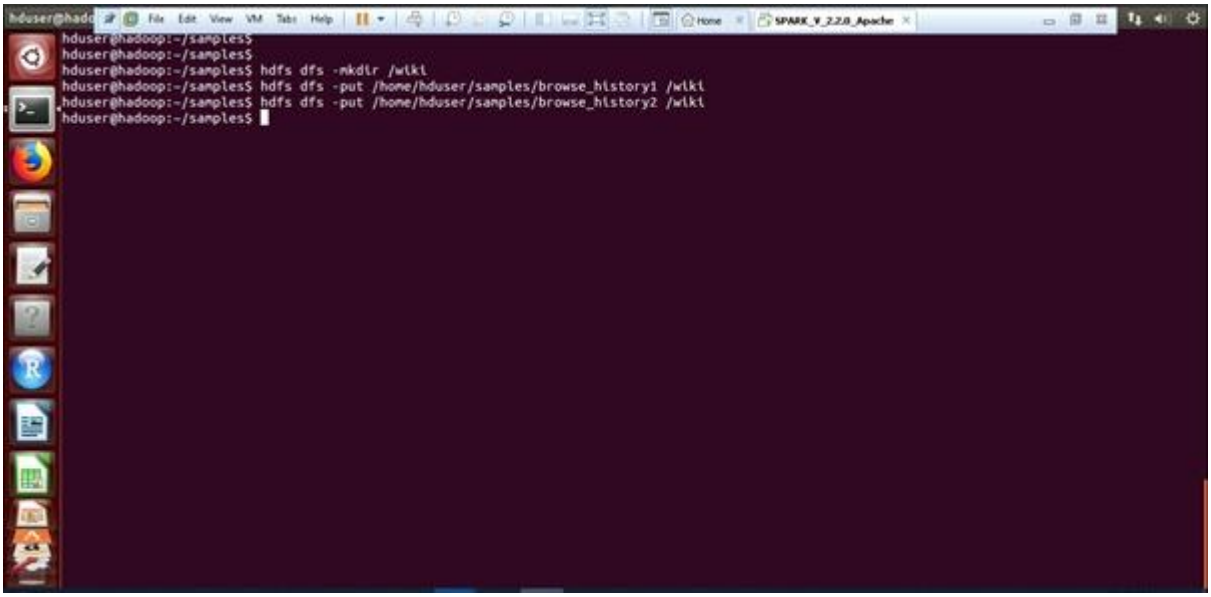
Create a directory named 'wiki' on HDFS. Place both the above files into the directory 'wiki' on HDFS. Create an External table named 'wiki_data' in the directory named 'myData' in the home directory with the following structure.

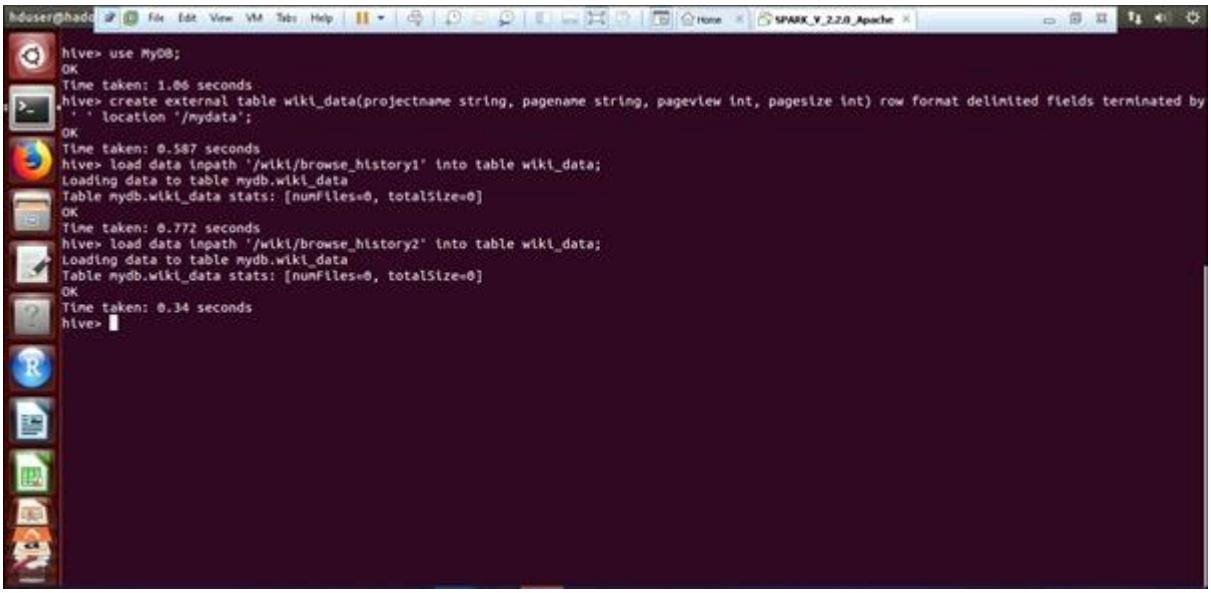
projectname	STRING
pagename	STRING
pageview	INT
pagesize	INT

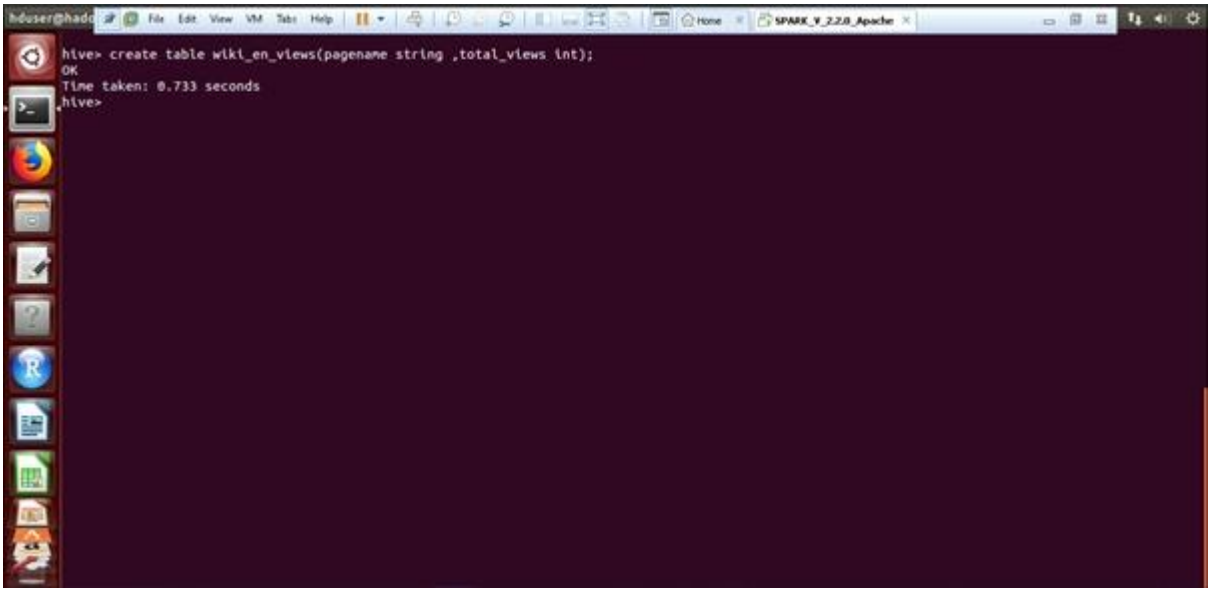
Now create an internal table named 'wiki_en_views' that has the total views for every page with the following structure,

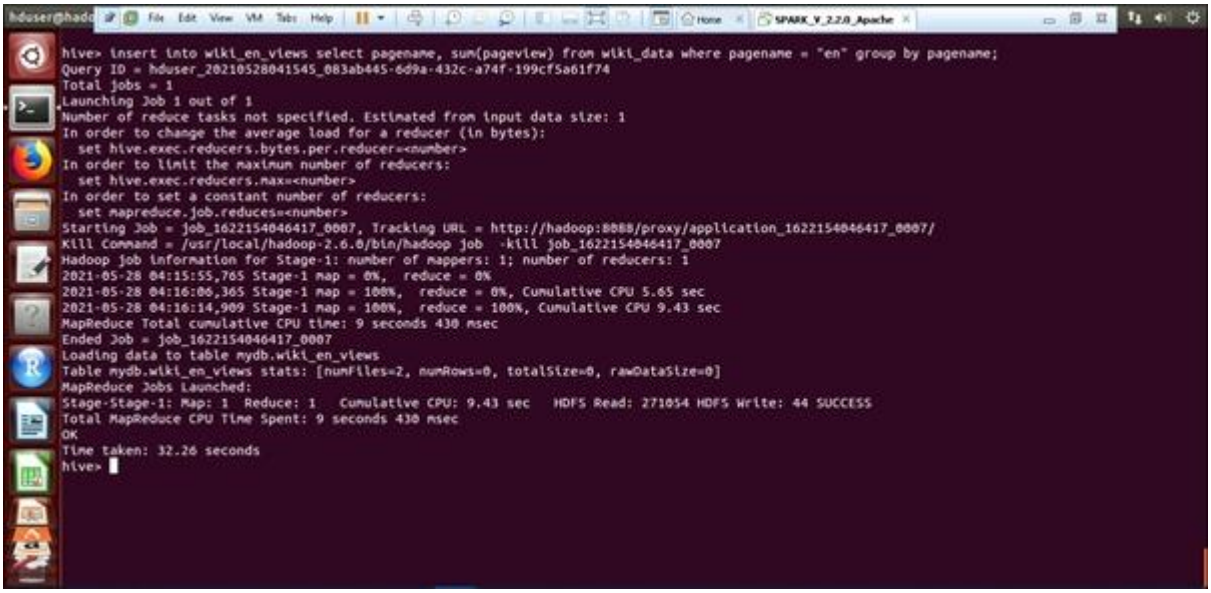
pagename	STRING
total_views	INT

Now load the 'wiki_en_view' table only for the projectname 'en' from the external table 'wiki'.









Requirement 9:

We have a file named “employee” in the samples directory of the home directory.

1. Create a Hive table named "EMP_DATA"
   Ans: **create table EMP_DATA (EMPNO int,ENAME varchar(20),JOB varchar(20),SAL int) partitioned by (DEPTNO int) row format delimited fields terminated by '\t';**

```
hive> create table EMP_DATA (empno int,ename varchar(20),job varchar(20),sal int,deptno int) partitioned by (deptno int) row format delimit
ed fields terminated by '\t';
FAILED: SemanticException [Error 10035]: Column repeated in partitioning columns
hive> create table EMP_DATA (empno int,ename varchar(20),job varchar(20),sal int) partitioned by (deptno int) row format delimited fields t
erminated by '\t';
OK
```

2. Alter the table to add the following 3 partitions
   a. Partition that has the employees of deptno 10
   Ans: **load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA partition(DEPTNO=10);**
   b. Partition that has the employees of deptno 20
   Ans: **load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA partition(DEPTNO=20);**

   c. Partition that has the employees of deptno 30
   Ans: **load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA partition(DEPTNO=30);**

```
hive> load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA
    > partition(DEPTNO=10);
Loading data to table mydb.emp_data partition (deptno=10)
Partition mydb.emp_data{deptno=10} stats: [numFiles=1, numRows=0, totalSize=401, rawDataSize=0]
OK
Time taken: 0.877 seconds
hive> load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA
    > partition(DEPTNO=20);
Loading data to table mydb.emp_data partition (deptno=20)
Partition mydb.emp_data{deptno=20} stats: [numFiles=1, numRows=0, totalSize=401, rawDataSize=0]
OK
Time taken: 0.798 seconds
hive> load data local inpath '/home/hduser/samples/employee.txt' into table EMP_DATA
    > partition(DEPTNO=30);
Loading data to table mydb.emp_data partition (deptno=30)
Partition mydb.emp_data{deptno=30} stats: [numFiles=1, numRows=0, totalSize=401, rawDataSize=0]
OK
Time taken: 0.937 seconds
```

```
7521    WARD    SALESMAN        1250    10
7566    JONES   MANAGER 2975    10
7654    MARTIN  SALESMAN        1400    10
7698    BLAKE   MANAGER 2850    10
7782    CLARK   MANAGER 2450    10
7788    SCOTT   ANALYST 3000    10
7839    KING    PRESIDENT       5000    10
7844    TURNER  SALESMAN        1500    10
7876    ADAMS   CLERK   1100    10
7900    JAMES   CLERK   950     10
7902    FORD    ANALYST 3000    10
7934    MILLER  CLERK   1300    10
NULL    ENAME   JOB     NULL    20
7369    SMITH   CLERK   800     20
7499    ALLEN   SALESMAN        1600    20
7521    WARD    SALESMAN        1250    20
7566    JONES   MANAGER 2975    20
7654    MARTIN  SALESMAN        1400    20
7698    BLAKE   MANAGER 2850    20
7782    CLARK   MANAGER 2450    20
7788    SCOTT   ANALYST 3000    20
7839    KING    PRESIDENT       5000    20
7844    TURNER  SALESMAN        1500    20
7876    ADAMS   CLERK   1100    20
7900    JAMES   CLERK   950     20
7902    FORD    ANALYST 3000    20
7934    MILLER  CLERK   1300    20
NULL    ENAME   JOB     NULL    30
7369    SMITH   CLERK   800     30
7499    ALLEN   SALESMAN        1600    30
7521    WARD    SALESMAN        1250    30
```

3. Write a query to display the employees from the 2nd partition that contains the employees of deptno 20.

Ans: **select * from EMP_DATA where DEPTNO=20;**

```
hive> select * from EMP_DATA where DEPTNO=20
    > ;
OK
NULL    ENAME   JOB     NULL    20
7369    SMITH   CLERK   800     20
7499    ALLEN   SALESMAN        1600    20
7521    WARD    SALESMAN        1250    20
7566    JONES   MANAGER 2975    20
7654    MARTIN  SALESMAN        1400    20
7698    BLAKE   MANAGER 2850    20
7782    CLARK   MANAGER 2450    20
7788    SCOTT   ANALYST 3000    20
7839    KING    PRESIDENT       5000    20
7844    TURNER  SALESMAN        1500    20
7876    ADAMS   CLERK   1100    20
7900    JAMES   CLERK   950     20
7902    FORD    ANALYST 3000    20
7934    MILLER  CLERK   1300    20
Time taken: 0.979 seconds, Fetched: 15 row(s)
hive>
```

Requirement 10:

We have a HIVE table created with the name "EMP" with the following columns

EMPNO	NUMERIC		(4)
ENAME	VARCHAR(		20)
MGRID NUMERIC		(4	)
DEPTNO	NUMERIC(	4)	
SAL	NUMERIC(	6)	
DEPTNO	NUMERIC(	2)	

We have another HIVE table created with the name "DEPT" with the following columns

DEPTNO	NUMERIC(	2)	
DNAME	VARCHAR(		20)
LOC	VARCHAR(	20)	

```
hive> create table EMP (
    > EMPNO INT,
    > ENAME VARCHAR(20),
    > JOB VARCHAR(20),
    > MGRID INT,
    > HIREDATE VARCHAR(20),
    > SAL INT,
    > COMM INT,
    > DEPTNO int) row format delimited fields terminated by ',';
OK
Time taken: 0.425 seconds
hive> describe EMP;
OK
empno                   int
ename                   varchar(20)
job                     varchar(20)
mgrid                   int
hiredate                varchar(20)
sal                     int
comm                    int
deptno                  int
Time taken: 0.157 seconds, Fetched: 8 row(s)
hive>
```

```
hive> select * from emp;
OK
7369    SMITH   CLERK7902       NULL    800     NULL    20      NULL
7499    ALLEN   SALESMAN        7698    20-FEB-81       1600    300     30
7521    WARD    SALESMAN        7698    22-FEB-81       1250    500     30
7566    JONES   MANAGER 7839    02-APR-81       2975    20      NULL
7654    MARTIN  SALESMAN        7698    28-SEP-81       1250    1400    30
7698    BLAKE   MANAGER 7839    01-MAY-81       2850    NULL    30
7782    CLARK   MANAGER 7839    09-JUN-81       2450    NULL    10
7788    SCOTT   ANALYST 7566    19-APR-87       3000    NULL    20
7839    KING    PRESIDENT       NULL    17-NOV-81       5000    NULL    10
7844    TURNER  SALESMAN        7698    08-SEP-81       1500    0       30
7876    ADAMS   CLERK   7788    23-MAY-87       1100    20      NULL
7900    JAMES   CLERK   7698    03-DEC-81       950     30      NULL
7902    FORD    ANALYST 7566    03-DEC-81       3000    20      NULL
7934    MILLER  CLERK   7782    23-JAN-82       1300    10      NULL
Time taken: 0.156 seconds, Fetched: 14 row(s)
```

```
hive> create table DEPT (
    > DEPTNO INT,
    > DNAME VARCHAR(20),
    > LOC VARCHAR(20))
    > row format delimited fields terminated by ',' tblproperties("skip.header.line.count"="1");
OK
Time taken: 0.339 seconds
hive> describe dept;
OK
deptno                  int
dname                   varchar(20)
loc                     varchar(20)
Time taken: 0.151 seconds, Fetched: 3 row(s)
hive>
```

Requirement is to create

1. A view that has only MANAGERs and a count of employees reporting to him with the following fields (DEPTNO Field is a common column available in both the tables, that should be used for JOIN)

   EMPNO, ENAME, JOB, SAL, DEPTNO, COUNT_OF_REPORTING_EMP

   Ans:

```
SELECT e.EMPNO,e.ENAME,e.JOB,e.SAL,e.DEPTNO,e1.COUNT_OF_REPORTING_EMP
FROM emp e,
  (
    SELECT
      MGRID,
      COUNT(*) COUNT_OF_REPORTING_EMP
    FROM
      emp
    GROUP BY
      MGRID
  ) e1 WHERE e.EMPNO = e1.MGRID;
```

```
hive> create view MANAGERS as
    > SELECT e.EMPNO,e.ENAME,e.JOB,e.SAL,e.DEPTNO,e1.COUNT_OF_REPORTING_EMP
    > FROM emp e,
    >     (
    >         SELECT
    >             MGRID,
    >             COUNT(*) COUNT_OF_REPORTING_EMP
    >         FROM
    >             emp
    >         GROUP BY
    >             MGRID
    >     ) e1 WHERE e.EMPNO = e1.MGRID;
OK
Time taken: 0.244 seconds
hive> select * from MANAGERS;
```

```
7566    JONES   MANAGER 2975    NULL    2
7698    BLAKE   MANAGER 2850    30      5
7782    CLARK   MANAGER 2450    10      1
7788    SCOTT   ANALYST 3000    20      1
7839    KING    PRESIDENT       5000    10      3
Time taken: 59.666 seconds, Fetched: 5 row(s)
```

2. A view that has the Employee details along with Managers details with the following structure (We don't have MGR_NAME Column. ENAME should be used in 2 different contexts – 1. EmpName and 2.MgrName)

   EMPNO, ENAME, JOB, MGRID, MGR_NAME, DEPTNO

   First step: create a manager_details table which contains employee details which are only manager.

```
OK
Time taken: 0.134 seconds
hive> create table manager_details (MGRID int,manager_name varchar(20))
    > row format delimited fields terminated by ',';
OK
Time taken: 0.131 seconds
hive> insert overwrite table manager_details
    > select distinct a.EMPNO,a.ENAME from EMP a, EMP b where a.EMPNO=b.MGRID;
```

```
hive> select * from manager_details;
OK
7566    JONES
7698    BLAKE
7782    CLARK
7788    SCOTT
7839    KING
Time taken: 0.372 seconds, Fetched: 5 row(s)
hive>
```

Ans:

create view EmpMgrDetails as select EMP.EMPNO,EMP.ENAME,EMP.JOB,manager_details.MGRID,manager_details.manager_name,EMP.DEPTNO from EMP inner join manager_details on EMP.MGRID=manager_details.MGRID;

```
hive> describe EmpMgrDetails;
OK
empno                   int
ename                   varchar(20)
job                     varchar(20)
mgrid                   int
manager_name            varchar(20)
deptno                  int
Time taken: 0.082 seconds, Fetched: 6 row(s)
```

Select * from EmpMgrDetails;

```
Total MapReduce CPU Time Spent: 3 seconds 510 msec
OK
7499    ALLEN   SALESMAN        7698    BLAKE   30
7521    WARD    SALESMAN        7698    BLAKE   30
7566    JONES   MANAGER 7839    KING    NULL
7654    MARTIN  SALESMAN        7698    BLAKE   30
7698    BLAKE   MANAGER 7839    KING    30
7782    CLARK   MANAGER 7839    KING    10
7788    SCOTT   ANALYST 7566    JONES   20
7844    TURNER  SALESMAN        7698    BLAKE   30
7876    ADAMS   CLERK   7788    SCOTT   NULL
7900    JAMES   CLERK   7698    BLAKE   NULL
7902    FORD    ANALYST 7566    JONES   NULL
7934    MILLER  CLERK   7782    CLARK   NULL
Time taken: 25.879 seconds, Fetched: 12 row(s)
hive> describe EmpMgrDetails;
OK
```

PIG Script

Requirement 11:

We have a file named "empDetails" in the samples directory of the home directory which is given below.

Write a PIG Script to count the number of each word in the file empDetails.

Ans:

```
lines = LOAD '/home/hduser/samples/empDetails.txt' AS (line:chararray);
DUMP lines;
```

```
o process : 1
(SMITH with the employee id 7369 is working as CLERK with a salary 800 in department no 20)
(ALLEN with the employee id 7499 is working as SALESMAN with a salary 1600 in department no 30)
(WARD with the employee id 7521 is working as SALESMAN with a salary 1250 in department no 30)
(JONES with the employee id 7566 is working as MANAGER with a salary 2975 in department no 20)
(MARTIN with the employee id 7654 is working as SALESMAN with a salary 1400 in department no 30)
(BLAKE with the employee id 7698 is working as MANAGER with a salary 2850 in department no 30)
(CLARK with the employee id 7782 is working as MANAGER with a salary 2450 in department no 10)
(SCOTT with the employee id 7788 is working as ANALYST with a salary 3000 in department no 20)
(KING with the employee id 7839 is working as PRESIDENT with a salary 5000 in department no 10)
(TURNER with the employee id 7844 is working as SALESMAN with a salary 1500 in department no 30)
(ADAMS with the employee id 7876 is working as CLERK with a salary 1100 in department no 20)
(JAMES with the employee id 7900 is working as CLERK with a salary 950 in department no 30)
(FORD with the employee id 7902 is working as ANALYST with a salary 3000 in department no 20)
(MILLER with the employee id 7934 is working as CLERK with a salary 1300 in department no 10)
()
```

```
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group,COUNT(words);
DUMP wordcount;
```

```
(7782,1)
(7788,1)
(7839,1)
(7844,1)
(7876,1)
(7900,1)
(7902,1)
(7934,1)
(FORD,1)
(KING,1)
(WARD,1)
(with,28)
(ADAMS,1)
(ALLEN,1)
(BLAKE,1)
(CLARK,1)
(CLERK,4)
(JAMES,1)
(JONES,1)
(SCOTT,1)
(SMITH,1)
(MARTIN,1)
(MILLER,1)
(TURNER,1)
(salary,14)
(ANALYST,2)
(MANAGER,3)
(working,14)
(SALESMAN,4)
(employee,14)
(PRESIDENT,1)
(department,14)
(,0)
grunt>
```

Requirement 12:

We have the following student data available in Samples directory of the Home directory with the following data,

student.txt

Write a PIG Script which is

1. to display the total marks of each student

Ans:

```
student = LOAD '/home/hduser/samples/student.txt'
  USING PigStorage(',')
  as ( id:int,name:chararray,subject:chararray,marks:int);

DUMP student;
```

```
(1000,RAJ,MATHS,56)
(1000,RAJ,PHYSICS,66)
(1000,RAJ,CHEMISTRY,76)
(1001,AMIT,MATHS,96)
(1001,AMIT,PHYSICS,86)
(1001,AMIT,CHEMISTRY,91)
(1002,DINESH,MATHS,82)
(1002,DINESH,PHYSICS,81)
(1002,DINESH,CHEMISTRY,84)
(1003,MAHESH,MATHS,77)
(1003,MAHESH,PHYSICS,78)
(1003,MAHESH,CHEMISTRY,79)
(1004,DAVID,MATHS,86)
(1004,DAVID,PHYSICS,86)
(1004,DAVID,CHEMISTRY,81)
grunt>
```

```
s_grouped = GROUP student BY name;
t_marks = FOREACH s_grouped GENERATE group,SUM(student.marks);
DUMP t_marks;
```

```
 paths to process : 1
(RAJ,198)
(AMIT,273)
(DAVID,253)
(DINESH,247)
(MAHESH,234)
grunt>
```

2. to display the total marks scored in every subject.

ANS:

sub_grouped = GROUP student BY subject;

t_scored = FOREACH sub_grouped GENERATE group, SUM(student.marks);

DUMP t_scored;

```
 paths to process : 1
(MATHS,397)
(PHYSICS,397)
(CHEMISTRY,411)
grunt>
```

Requirement 13:

We have the EMP data which has empno, ename, job, sal and deptno fields available in HDFS(Emp file is embedded here).

Write a PigScript to display the department wise employee details s for every deptno available in the emp file (Hint : Use the GROUP Operator)

Ans:

```
emp = LOAD '/home/hduser/samples/emp.txt'
  USING PigStorage(',')
  as (empno:int,ename:chararray,job:chararray,sal:int,deptno:int);

DUMP emp;
```

```
o process : 1
2021-06-03 20:08:18,618 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total inpu
 paths to process : 1
(7369,SMITH,CLERK,800,20)
(7499,ALLEN,SALESMAN,1600,30)
(7521,WARD,SALESMAN,1250,30)
(7566,JONES,MANAGER,2975,20)
(7654,MARTIN,SALESMAN,1400,30)
(7698,BLAKE,MANAGER,2850,30)
(7782,CLARK,MANAGER,2450,10)
(7788,SCOTT,ANALYST,3000,20)
(7839,KING,PRESIDENT,5000,10)
(7844,TURNER,SALESMAN,1500,30)
(7876,ADAMS,CLERK,1100,20)
(7900,JAMES,CLERK,950,30)
(7902,FORD,ANALYST,3000,20)
(7934,MILLER,CLERK,1300,10)
(,,,,)
grunt>
```

dept_emp = GROUP emp BY deptno;

DUMP dept_emp;

```
 paths to process : 1
(10,{(7782,CLARK,MANAGER,2450,10),(7934,MILLER,CLERK,1300,10),(7839,KING,PRESIDENT,5000,10)})
(20,{(7369,SMITH,CLERK,800,20),(7902,FORD,ANALYST,3000,20),(7876,ADAMS,CLERK,1100,20),(7788,SCOTT,ANALYST,3000,2
0),(7566,JONES,MANAGER,2975,20)})
(30,{(7900,JAMES,CLERK,950,30),(7698,BLAKE,MANAGER,2850,30),(7654,MARTIN,SALESMAN,1400,30),(7521,WARD,SALESMAN,1
250,30),(7844,TURNER,SALESMAN,1500,30),(7499,ALLEN,SALESMAN,1600,30)})
(,{(,,,,)})
grunt>
```

Requirement 14:

We have the EMP data and DEPT data available in HDFS as follows,


emp.txt


dept.txt

Write a PigScript to display the department details and the list of employees working in the department for every deptno .

ANS:

```
emp1 = LOAD '/home/hduser/samples/EMP.txt'
  USING PigStorage(',')
  as
(empno:int,ename:chararray,job:chararray,MGR:int,HIREDATE:chararray,sal:int,COMM:int,deptno:int
);
DUMP emp1
```

```
(7369,SMITH,CLERK7902,,800,,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
grunt>
```

```
;
```

```
dept = LOAD '/home/hduser/samples/dept.txt'
  USING PigStorage(',')
  as (DEPTNO:int,DNAME:chararray,LOC:chararray);
DUMP dept;
```

```
ized
2021-06-03 20:24:12,094 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to proces
 : 1
2021-06-03 20:24:12,094 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths t
 process : 1
(10,ACCOUNTING,NEWYORK)
(20,RESEARCH,DALLAS)
(30,SALES,CHICAGO)
(40,OPERATIONS,BOSTON)
(,,)
grunt>
```

emp1_dept = JOIN emp1 BY deptno, dept BY DEPTNO;

result = FOREACH emp1_dept GENERATE $0,$8,$9,$10;
DUMP result;

```
ized
2021-06-03 20:28:13,294 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to proces
 : 1
2021-06-03 20:28:13,294 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths t
 process : 1
(7839,KING,10,ACCOUNTING,NEWYORK)
(7782,CLARK,10,ACCOUNTING,NEWYORK)
(7788,SCOTT,20,RESEARCH,DALLAS)
(7698,BLAKE,30,SALES,CHICAGO)
(7654,MARTIN,30,SALES,CHICAGO)
(7844,TURNER,30,SALES,CHICAGO)
(7521,WARD,30,SALES,CHICAGO)
(7499,ALLEN,30,SALES,CHICAGO)
grunt>
```

Requirement 15:

We have the CUSTOMER data and ORDERS data available in HDFS as follows,


customer.txt


orders.txt

Write a Pig Script to

1. display the customer details along with the order details of that customer.

ANS:

```
customer = LOAD '/home/hduser/samples/customer.txt'
  USING PigStorage(',')
  as (custid:int,custname:chararray,age:int,city:chararray,salary:int);
DUMP customer;
```

```
2021-06-03 20:29:26,811 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input path
 process : 1
(1,Ramesh,32,Ahmedabad,2000)
(2,Khilan,25,Delhi,1500)
(3,kaushik,23,Kota,2000)
(4,Chaitali,25,Mumbai,6500)
(5,Hardik,27,Bhopal,8500)
(6,Komal,22,MP,4500)
(7,Muffy,24,Indore,10000)
(,,,,)
grunt>
```

orders = LOAD '/home/hduser/samples/order.txt'
USING PigStorage(',')
as (orderid:int,orderdate:chararray,custid:int,amount:int);
DUMP orders;

```
paths to process : 1
2021-06-03 20:54:00,759 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil -
l input paths to process : 1
(102,2009-10-08 00:00:00,3,3000)
(100,2009-10-08 00:00:00,3,1500)
(101,2009-11-20 00:00:00,2,1560)
(103,2008-05-20 00:00:00,4,2060)
grunt>
```

cust_order = JOIN customer BY custid, orders BY custid;
DUMP cust_order;

```
2021-06-03 20:55:00,996 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil -
l input paths to process : 1
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
grunt>
```

2. display all the customer details along with the order details of that customer, though the customer has not ordered anything.

Ans:

cust_order_left = JOIN customer BY custid LEFT OUTER, orders BY custid;
DUMP cust_order_left;

```
2021-06-03 20:55:52,631 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total inpu
paths to process : 1
2021-06-03 20:55:52,631 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - To
l input paths to process : 1
(1,Ramesh,32,Ahmedabad,2000,,,,)
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,,,,)
(6,Komal,22,MP,4500,,,,)
(7,Muffy,24,Indore,10000,,,,)
```

Requirement 16:

We have the employee details available in 2 different files named EMP1 and EMP2 in HDFS. The requirement is combine all the employee details from EMP1 and EMP2.


emp1.txt


emp2.txt

Write a Pig Script to combine the employee details from EMP1 and EMP2.

Ans:

```
e1 = LOAD '/home/hduser/samples/emp1.txt'
  USING PigStorage(',')
  as
(empno:int,ename:chararray,job:chararray,MGR:int,HIREDATE:chararray,sal:int,COMM:int,deptno:int
);
DUMP e1;
```

```
l input paths to process : 1
(7369,SMITH,CLERK7902,,800,,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(,,,,,,,)
grunt>
```

e2 = LOAD '/home/hduser/samples/emp2.txt'
USING PigStorage(',')
as
(empno:int,ename:chararray,job:chararray,MGR:int,HIREDATE:chararray,sal:int,COMM:int,deptno:int);
DUMP e2;

```
2021-06-03 20:58:16,683 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Tot
l input paths to process : 1
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(,,,,,,,)
grunt>
```

e1_e2 = CROSS e1, e2;
DUMP e1_e2;

```
2021-06-03 20:59:24,676 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l input paths to process : 1
(,,,,,,,,,,,,,,,)
(,,,,,,,,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(,,,,,,,,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(,,,,,,,,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(,,,,,,,,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(,,,,,,,,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(,,,,,,,,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,,,,,,,,)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,,,,,,,,)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30,,,,,,,,)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,,,,,,,,)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,,,,,,,,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7369,SMITH,CLERK7902,,800,,20,,,,,,,,,)
(7369,SMITH,CLERK7902,,800,,20,,7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
(7369,SMITH,CLERK7902,,800,,20,,7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7369,SMITH,CLERK7902,,800,,20,,7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7369,SMITH,CLERK7902,,800,,20,,7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7369,SMITH,CLERK7902,,800,,20,,7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7369,SMITH,CLERK7902,,800,,20,,7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
grunt>
```

Requirement 17:

We have the EMP data available in HDFS as follows,

Write a PigScript to split the data into two, one has the list of employees whose salary is less than 2800 and the second one the list of employees whose salary is greater than or equal to 2800 (Hint : Use the SPLIT Operator)

Ans:

```
emp1 = LOAD '/home/hduser/samples/EMP.txt'
  USING PigStorage(',')
  as
(empno:int,ename:chararray,job:chararray,MGR:int,HIREDATE:chararray,sal:int,COMM:int,deptno:int
);
DUMP emp1;
```

```
paths to process : 1
2021-06-03 21:02:59,253 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Tot
l input paths to process : 1
(7369,SMITH,CLERK7902,,800,,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
grunt>
```

```
SPLIT emp1 into emp11 if (sal<2800), emp12 if (sal>=2800);
DUMP emp11;
```

```
2021-06-03 21:03:46,791 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input
paths to process : 1
2021-06-03 21:03:46,791 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Tota
l input paths to process : 1
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30)
(7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10)
(7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
grunt>
```

DUMP emp12;

```
ady been initialized
2021-06-03 21:03:58,703 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input
paths to process : 1
2021-06-03 21:03:58,703 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Tota
l input paths to process : 1
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
grunt>
```

Requirement 18:

We have the EMP data available in HDFS as follows,

emp.txt

Write a Pig Script to display the TOP FIVE SALARIED employees. (Hint : Use ORDER BY and LIMIT operators)

Ans:

emp1 = LOAD '/home/hduser/samples/EMP.txt'

USING PigStorage(',')
as
(empno:int,ename:chararray,job:chararray,MGR:int,HIREDATE:chararray,sal:int,COMM:int,deptno:int);
DUMP emp1;

```
paths to process : 1
2021-06-03 21:02:59,253 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Tot
l input paths to process : 1
(7369,SMITH,CLERK7902,,800,,20,)
(7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30)
(7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
(7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30)
(7876,ADAMS,CLERK,7788,23-MAY-87,1100,20,)
(7900,JAMES,CLERK,7698,03-DEC-81,950,30,)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7934,MILLER,CLERK,7782,23-JAN-82,1300,10,)
grunt>
```

order_by_emp1 = ORDER emp1 BY sal DESC;
limit_emp1 = LIMIT order_by_emp1 5;

```
otal input paths to process : 1
2021-06-03 21:17:00,865 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRed
Util - Total input paths to process : 1
(7839,KING,PRESIDENT,,17-NOV-81,5000,,10)
(7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20)
(7902,FORD,ANALYST,7566,03-DEC-81,3000,20,)
(7566,JONES,MANAGER,7839,02-APR-81,2975,20,)
(7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30)
grunt>
```

Important Instructions:

- Please read the document thoroughly before you code.
- Please do not change the Business Requirements.

Coverage:

5. HDFS
6. Map Reduce
7. HIVE
8. PIG Script

For all the hands-on, please consider the following,

UNIX User name: training

Home Directory: /home/training

SPARK

Requirement 1:

We have the fruits details available in a CSV file named fruits.txt with column names in the 1st line. Write a Scala program using Spark to display the content of the file skipping the column headers.

fruits.txt

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
val sc = SparkContext.getOrCreate()
val dataFrame  = spark.read.format("csv").option("header","true").option("inferSchema",
"true").load("/FileStore/tables/fruits.text")
print(dataFrame)
dataFrame.collect()
```

1	1	Apple
2	2	Banana
3	3	Orange
4	4	Grapes

Showing all 4 rows.

Requirement 2:

We have the four CSV files named fruits.txt, animal.txt, city.txt and country.txt in a directory named d:\demo\spark, with column names appearing as the 1st line. Write a Scala program using Spark to read all the files in the directory and display the content of the files skipping the column headers.


```
val dF = spark.read.format("csv").option("header","true").option("inferSchema",
"true").load("/FileStore/tables/animal-1.text","/FileStore/tables/country-1.text","/FileStore/tables/fruits-
2.text","/FileStore/tables/city-1.text")
dF.collect()
dF.show()
```

```
+---+--------+
|  1| Monkey|
|  2| Donkey|
|  3|  Horse|
|  4|Gorilla|
|  1|  Apple|
|  2| Banana|
|  3| Orange|
|  4| Grapes|
|  1|  Texas|
|  2| Dallas|
|  3|  Miami|
|  4|Chicago|
|  1|     UK|
|  2|    USA|
|  3| Mexico|
|  4| Canada|
+---+--------+
```

Requirement 3:

We need to count the number of words that starts with the letter 'A'. To achieve this, we are supposed to write a Scala program using Spark. You are supposed to use RDD Transformation functions to implement this. Use the following Transformations functions -flatMap(), map(), filter() and reduceByKey() to achieve this.

File: Names.txt

Sample Data
Arpit
Sowmya
Jigyasa
Thiru
Mahesh
Arun
Saurav
Lalith

Keerthana
Nikita
Aiswarya
Nency
Kalyan

```
val data=sc.textFile("/FileStore/tables/Names-1.text")
data.collect()
val splitdata = data.flatMap(line => line.split(" "));
val output = splitdata.filter{word=> word.startsWith("A")}
output.collect()
val mapdata = output.map(word => (word,1));
mapdata.collect()
val count=mapdata.count()
```

```
val count=mapdata.count()

count: Long = 3
```

Requirement 4:

We have the order details available in orders.txt. The requirement is to count the number of orders placed by the customer. You are supposed to write a Scala program using Spark to display the number of orders placed by each customer.


```
val order=sc.textFile("/FileStore/tables/orders.text")
order.collect()
val order = spark.read.format("csv").option("header","true").option("inferSchema",
"true").load("/FileStore/tables/orders.text")
order.registerTempTable("order")
sqlContext.sql("SELECT CustId, COUNT(*) AS cnt FROM order GROUP BY custId").show()
```

```
+---------+---+
|CustId|cnt|
+------+---+
|     3|  2|
|     4|  1|
|     2|  1|
+------+---+
```

Requirement 5:

Assume you are getting a two-letter state code in a file named data.txt and the requirement is to transform it to full state name, (for example AP to ANDRAPRADESH, TN to TAMILNADU etc.) by doing a lookup to reference mapping.

You need to use the broadcast variable to cache this lookup info (which is available in state.txt) on each machine and tasks use this cached info while executing the transformations.


data.txt


state.txt

```
val states =
Map(("BR","BIHAR"),("MH","MAHARASHTRA"),("TN","TAMILNADU"),("AP","ANDRAPRADESH"),("KL","KERALA"))
 val broadcastStates  = spark.sparkContext.broadcast(states)
 val data = Seq(
  ("Amit","Mishra","BR"),
  ("Nitin","Kulkarni","MH"),
  ("Ram","Kumar","TN"),
  ("Kesav","Prasad","AP"),
  ("Biju","Joseph","KL"),
  ("Ravi","Teja","AP")
 )
 val rdd = spark.sparkContext.parallelize(data)

 val rdd2  = rdd.map(f=>{
  val state = f._3
  val fullState = broadcastStates.value.get(state).get
  (f._1,f._2,fullState)
 })
```

```
println(rdd2.collect().mkString("\n"))
```

```
(Amit,Mishra,BIHAR)
(Nitin,Kulkarni,MAHARASHTRA)
(Ram,Kumar,TAMILNADU)
(Kesav,Prasad,ANDRAPRADESH)
(Biju,Joseph,KERALA)
(Ravi,Teja,ANDRAPRADESH)
states: scala.collection.immutable.Map[String,String] = Map(MH -> MAHARASHTRA, TN -> TAMILNADU, KL -> KERALA, BR -> BIHAR, AP -> ANDRAPRADESH)
broadcastStates: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,String]] = Broadcast(21)
data: Seq[(String, String, String)] = List((Amit,Mishra,BR), (Nitin,Kulkarni,MH), (Ram,Kumar,TN), (Kesav,Prasad,AP), (Biju,Joseph,KL), (Ravi,Teja,AP))
rdd: org.apache.spark.rdd.RDD[(String, String, String)] = ParallelCollectionRDD[26] at parallelize at command-2826642787694273:11
rdd2: org.apache.spark.rdd.RDD[(String, String, String)] = MapPartitionsRDD[27] at map at command-2826642787694273:13
```

```
+-----+--------+---+
| Amit|  Mishra| BR|
+-----+--------+---+
|Nitin|Kulkarni| MH|
|  Ram|   Kumar| TN|
|Kesav|  Prasad| AP|
| Biju|  Joseph| KL|
| Ravi|    Teja| AP|
+-----+--------+---+

broadcastStates: org.apache.spark.broadcast.Broadcast[Array[String]] = Broadcast(20)
```

Requirement 6:

Introduce an array with few values. Write a Scala program to perform the total value of all elements of the array. You need to implement this problem by using a Named Accumulator.

```
val data=Array(1, 2, 3, 4)
val accum = sc.longAccumulator("My Accumulator")
sc.parallelize(data).foreach(x => accum.add(x))
accum.value
```

```
data: Array[Int] = Array(1, 2, 3, 4)
accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 522, name: Some(My Accumulator), value: 10)
res23: Long = 10
```

Requirement 7:

We have the population of few cities available in a sequence as follows,

Seq (("Chennai", "20000000"), ("Mumbai", "50000000"), ("Delhi", "40000000"))

Write a Scala program using Spark to convert the above sequence into a Data Frame with two columns (City and humanCount).

```
val data =Seq (("Chennai", "20000000"), ("Mumbai", "50000000"), ("Delhi", "40000000"))
val columns = Seq("city","humanCount")
 import spark.sqlContext.implicits._
 val df = data.toDF(columns:_*)
```

```
data: Seq[(String, String)] = List((Chennai,20000000), (Mumbai,50000000), (Delhi,40000000))
columns: Seq[String] = List(city, humanCount)
import spark.sqlContext.implicits._
df: org.apache.spark.sql.DataFrame = [city: string, humanCount: string]
```

```
df.show()
```

```
+-------+----------+
|   city|humanCount|
+-------+----------+
|Chennai|  20000000|
| Mumbai|  50000000|
|  Delhi|  40000000|
+-------+----------+
```

Requirement 8:

We have the skillset of the candidates available in a sequence as follows,

```
Seq(
  Row("Amit,,Mishra",List("Java","Scala","C++"),"UP"),
  Row("Prabhu,Ram,",List("Spark","Java","C++"),"TN"),
  Row("Ramesh,Kumar",List("CSharp","VB"),"TN")
 )
```

You need to write a Scala program using Spark by converting the array of strings (skillset) into string column in a Dataframe.

```
import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType( Array(
          StructField("Name", StringType,true),
          StructField("Skillset", ArrayType(StringType),true),
          StructField("State", StringType,true),
        ))
```

```
val data =Seq(
   Row("Amit,,Mishra",List("Java","Scala","C++"),"UP"),
   Row("Prabhu,Ram,",List("Spark","Java","C++"),"TN"),
   Row("Ramesh,Kumar",List("CSharp","VB"),"TN"))
var dfFromData3  = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData3.show()
```

```
+------------+------------------+-----+
|        Name|          Skillset|State|
+------------+------------------+-----+
|Amit,,Mishra|[Java, Scala, C++]|   UP|
| Prabhu,Ram,|[Spark, Java, C++]|   TN|
|Ramesh,Kumar|      [CSharp, VB]|   TN|
+------------+------------------+-----+

import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
schema: org.apache.spark.sql.types.StructType = StructType(StructField(Name,StringType,true), StructField(Skillset,ArrayType(StringType,true),true), StructField(State,StringType,true))
data: Seq[org.apache.spark.sql.Row] = List([Amit,,Mishra,List(Java, Scala, C++),UP], [Prabhu,Ram,,List(Spark, Java, C++),TN], [Ramesh,Kumar,List(CSharp, VB),TN])
dfFromData3: org.apache.spark.sql.DataFrame = [Name: string, Skillset: array<string> ... 1 more field]
```

Requirement 9:

Seq(

ROW(7369,"SMITH","CLERK",800,20)

ROW(7499,"ALLEN","SALESMAN",1600,30)

ROW(7521,"WARD","SALESMAN",1250,30)

ROW(7566,"JONES","MANAGER",2975,20)

ROW(7654,"MARTIN","SALESMAN",1400,30)

ROW(7698,"BLAKE",MANAGER",2850,30)

)

You need to write a Scala program using Spark to perform

1. Create a Data Frame from the above Sequence with the following column list
    a. EMPNO
    b. ENAME
    c. JOB
    d. SALARY
    e. DEPTNO
2. Update the salary column by adding $100
3. Create a new column named “Bonus” which is 20% of the salary

```
import org.apache.spark.sql.types.{StringType,  StructField,  StructType,  ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType(  Array(
```

```
        StructField("Empno",  IntegerType,true),
        StructField("Ename",StringType,true),
        StructField("Job",  StringType,true),
        StructField("Sal",  IntegerType,true),
        StructField("Deptno",  IntegerType,true),
      ))
val data =Seq(
  Row(7369,"SMITH","CLERK",800,20),
  Row(7499,"ALLEN","SALESMAN",1600,30),
  Row(7521,"WARD","SALESMAN",1250,30),
  Row(7566,"JONES","MANAGER",2975,20),
  Row(7654,"MARTIN","SALESMAN",1400,30),
  Row(7698,"BLAKE","MANAGER",2850,30)
)
var dfFromData  = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData.show()
```

```
+-----+------+--------+----+------+
|Empno| Ename|     Job| Sal|Deptno|
+-----+------+--------+----+------+
| 7369| SMITH|   CLERK| 800|    20|
| 7499| ALLEN|SALESMAN|1600|    30|
| 7521|  WARD|SALESMAN|1250|    30|
| 7566| JONES| MANAGER|2975|    20|
| 7654|MARTIN|SALESMAN|1400|    30|
| 7698| BLAKE| MANAGER|2850|    30|
+-----+------+--------+----+------+
```

```
dfFromData.createOrReplaceTempView("EmpDetails")
dfFromData.withColumn("Sal",dfFromData("Sal")+100).show(false)
```

```
+-----+------+--------+----+------+
|Empno|Ename |Job     |Sal |Deptno|
+-----+------+--------+----+------+
|7369 |SMITH |CLERK   |900 |20    |
|7499 |ALLEN |SALESMAN|1700|30    |
|7521 |WARD  |SALESMAN|1350|30    |
|7566 |JONES |MANAGER |3075|20    |
|7654 |MARTIN|SALESMAN|1500|30    |
|7698 |BLAKE |MANAGER |2950|30    |
+-----+------+--------+----+------+
```

```
dfFromData.withColumn("Bonus",dfFromData("Sal")*0.20).show(false)
```

```
+-----+------+---------+----+------+-----+
|Empno|Ename |Job      |Sal |Deptno|Bonus|
+-----+------+---------+----+------+-----+
|7369 |SMITH |CLERK    |800 |20    |160.0|
|7499 |ALLEN |SALESMAN|1600|30    |320.0|
|7521 |WARD  |SALESMAN|1250|30    |250.0|
|7566 |JONES |MANAGER |2975|20    |595.0|
|7654 |MARTIN|SALESMAN|1400|30    |280.0|
|7698 |BLAKE |MANAGER |2850|30    |570.0|
+-----+------+---------+----+------+-----+
```

Requirement 10:

Seq(

ROW(7369,"SMITH","CLERK",800,"Operations")
ROW(7499,"ALLEN","SALESMAN",1600,"Marketing")
ROW(7521,"WARD","SALESMAN",1250, "Marketing")
ROW(7566,"JONES","MANAGER",2975, "Operations")
ROW(7654,"MARTIN","SALESMAN",1400, "Marketing")
ROW(7698,"BLAKE",MANAGER",2850, "Marketing")

)

You need to write a Scala program using Spark to perform

1. Create a Data Frame from the above Sequence
2. Display the department wise sum of salary.
3. Display the department wise job wise sum of salary.
4. Display the department wise sum of salary where the sum of salary >= 5000.

```
import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType( Array(
          StructField("Empno", IntegerType,true),
          StructField("Ename",StringType,true),
          StructField("Job", StringType,true),
          StructField("Sal", IntegerType,true),
          StructField("Dept", StringType,true),
        ))
val data =Seq(
```

```
  Row(7369,"SMITH","CLERK",800,"Operations"),
  Row(7499,"ALLEN","SALESMAN",1600,"Marketing"),
  Row(7521,"WARD","SALESMAN",1250, "Marketing"),
  Row(7566,"JONES","MANAGER",2975, "Operations"),
  Row(7654,"MARTIN","SALESMAN",1400, "Marketing"),
  Row(7698,"BLAKE","MANAGER",2850, "Marketing"),

)
var dfFromData = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData.show()
```

```
+-----+------+--------+----+----------+
|Empno| Ename|     Job| Sal|      Dept|
+-----+------+--------+----+----------+
| 7369| SMITH|   CLERK| 800|Operations|
| 7499| ALLEN|SALESMAN|1600| Marketing|
| 7521|  WARD|SALESMAN|1250| Marketing|
| 7566| JONES| MANAGER|2975|Operations|
| 7654|MARTIN|SALESMAN|1400| Marketing|
| 7698| BLAKE| MANAGER|2850| Marketing|
+-----+------+--------+----+----------+
```

```
dfFromData.createOrReplaceTempView("EmpDept")
spark.sql("Select Dept,sum(Sal) from EmpDept group by Dept").show()
```

```
+----------+---------+
|      Dept|sum(Sal)|
+----------+---------+
|Operations|     3775|
| Marketing|     7100|
+----------+---------+
```

```
spark.sql("Select Dept,Job,sum(Sal) from EmpDept group by Dept,Job").show()
```

```
+----------+--------+---------+
|      Dept|     Job|sum(Sal)|
+----------+--------+---------+
|Operations|   CLERK|      800|
| Marketing|SALESMAN|     4250|
|Operations| MANAGER|     2975|
| Marketing| MANAGER|     2850|
+----------+--------+---------+
```

```
spark.sql("Select Dept,sum(Sal) from EmpDept group by Dept having(sum(sal)>=5000)").show()
```

```
+---------+--------+
|     Dept|sum(Sal)|
+---------+--------+
|Marketing|    7100|
```

Requirement 11:

Seq(

ROW(7369,"SMITH","CLERK", 800, 20)
ROW(7499,"ALLEN","SALESMAN", 1600, 30)
ROW(7521,"WARD","SALESMAN",1250,30)
ROW(7566,"JONES","MANAGER",2975,20)
ROW(7654,"MARTIN","SALESMAN",1400,30)
ROW(7698,"BLAKE",MANAGER",2850,30)

)

Seq(

ROW(10,"ACCOUNTING","NEWYORK"),
ROW(20,"RESEARCH","DALLAS"),
ROW(30,"SALES","CHICAGO"),
ROW(40,"OPERATIONS","BOSTON")

)

You need to write a Scala program using Spark to perform

1. Combine EMP and DEPT data frames based on the Key column DepdId
2. Combine EMP and DEPT data frames based on the Key column DepdId but should include all the employee details
3. Combine EMP and DEPT data frames based on the Key column DepdId but should include all the dept details

```
import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType( Array(
        StructField("Empno", IntegerType,true),
        StructField("Ename",StringType,true),
        StructField("Job", StringType,true),
        StructField("Sal", IntegerType,true),
        StructField("DeptId", IntegerType,true),
```

```
        ))
val data =Seq(
   Row(7369,"SMITH","CLERK",800,20),
   Row(7499,"ALLEN","SALESMAN",1600,30),
   Row(7521,"WARD","SALESMAN",1250,30),
   Row(7566,"JONES","MANAGER",2975,20),
   Row(7654,"MARTIN","SALESMAN",1400,30),
   Row(7698,"BLAKE","MANAGER",2850,30)
)
var dfFromData1  = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData1.show()
```

```
+-----+------+--------+----+------+
|Empno| Ename|     Job| Sal|DeptId|
+-----+------+--------+----+------+
| 7369| SMITH|   CLERK| 800|    20|
| 7499| ALLEN|SALESMAN|1600|    30|
| 7521|  WARD|SALESMAN|1250|    30|
| 7566| JONES| MANAGER|2975|    20|
| 7654|MARTIN|SALESMAN|1400|    30|
| 7698| BLAKE| MANAGER|2850|    30|
+-----+------+--------+----+------+
```

```
import org.apache.spark.sql.types.{StringType,  StructField,  StructType,  ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema= StructType(  Array(
          StructField("DeptId",   IntegerType,true),
          StructField("Dept",StringType,true),
          StructField("Loc",  StringType,true),
        ))
val data =Seq(
   Row(10,"ACCOUNTING","NEWYORK"),
   Row(20,"RESEARCH","DALLAS"),
   Row(30,"SALES","CHICAGO"),
   Row(40,"OPERATIONS","BOSTON"))
var dfFromData2  = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData2.show()
```

```
+------+----------+-------+
|DeptId|      Dept|    Loc|
+------+----------+-------+
|    10|ACCOUNTING|NEWYORK|
|    20|  RESEARCH| DALLAS|
|    30|     SALES|CHICAGO|
|    40|OPERATIONS| BOSTON|
+------+----------+-------+
```

```
dfFromData1.join(dfFromData2,dfFromData1("DeptId") === dfFromData2("DeptId"),"inner").show(false)
```

```
+-----+------+--------+----+------+------+--------+-------+
|Empno|Ename |Job     |Sal |DeptId|DeptId|Dept    |Loc    |
+-----+------+--------+----+------+------+--------+-------+
|7369 |SMITH |CLERK   |800 |20    |20    |RESEARCH|DALLAS |
|7566 |JONES |MANAGER |2975|20    |20    |RESEARCH|DALLAS |
|7499 |ALLEN |SALESMAN|1600|30    |30    |SALES   |CHICAGO|
|7521 |WARD  |SALESMAN|1250|30    |30    |SALES   |CHICAGO|
|7654 |MARTIN|SALESMAN|1400|30    |30    |SALES   |CHICAGO|
|7698 |BLAKE |MANAGER |2850|30    |30    |SALES   |CHICAGO|
+-----+------+--------+----+------+------+--------+-------+
```

```
dfFromData1.join(dfFromData2,dfFromData1("DeptId") ===
dfFromData2("DeptId"),"leftouter").show(false)
```

```
+-----+------+--------+----+------+------+--------+-------+
|Empno|Ename |Job     |Sal |DeptId|DeptId|Dept    |Loc    |
+-----+------+--------+----+------+------+--------+-------+
|7369 |SMITH |CLERK   |800 |20    |20    |RESEARCH|DALLAS |
|7566 |JONES |MANAGER |2975|20    |20    |RESEARCH|DALLAS |
|7499 |ALLEN |SALESMAN|1600|30    |30    |SALES   |CHICAGO|
|7521 |WARD  |SALESMAN|1250|30    |30    |SALES   |CHICAGO|
|7654 |MARTIN|SALESMAN|1400|30    |30    |SALES   |CHICAGO|
|7698 |BLAKE |MANAGER |2850|30    |30    |SALES   |CHICAGO|
+-----+------+--------+----+------+------+--------+-------+
```

```
dfFromData1.join(dfFromData2,dfFromData1("DeptId") ===
dfFromData2("DeptId"),"rightouter").show(false)
```

```
+-----+------+---------+----+------+------+----------+-------+
|Empno|Ename |Job      |Sal |DeptId|DeptId|Dept      |Loc    |
+-----+------+---------+----+------+------+----------+-------+
|null |null  |null     |null|null  |10    |ACCOUNTING|NEWYORK|
|7369 |SMITH |CLERK    |800 |20    |20    |RESEARCH  |DALLAS |
|7566 |JONES |MANAGER  |2975|20    |20    |RESEARCH  |DALLAS |
|7499 |ALLEN |SALESMAN|1600|30    |30    |SALES     |CHICAGO|
|7521 |WARD  |SALESMAN|1250|30    |30    |SALES     |CHICAGO|
|7654 |MARTIN|SALESMAN|1400|30    |30    |SALES     |CHICAGO|
|7698 |BLAKE |MANAGER  |2850|30    |30    |SALES     |CHICAGO|
|null |null  |null     |null|null  |40    |OPERATIONS|BOSTON |
+-----+------+---------+----+------+------+----------+-------+
```

Requirement 12:

Seq(

ROW(7369,"SMITH","CLERK",800,20)

ROW(7499,"ALLEN","SALESMAN",1600,30)

ROW(7521,"WARD","SALESMAN",1250,30)

ROW(7566,"JONES","MANAGER",2975,20)

ROW(7654,"MARTIN","SALESMAN",1400,30)

ROW(7698,"BLAKE",MANAGER",2850,30)

)

You need to write a Scala program using Spark to perform

1. Create a Data Frame from the above Sequence
2. Assign rank no for each employee based on the salary for every department using Windowing function.

```
import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType( Array(
          StructField("Empno", IntegerType,true),
          StructField("Ename",StringType,true),
          StructField("Job", StringType,true),
          StructField("Sal", IntegerType,true),
          StructField("DeptId", IntegerType,true),
        ))
val data =Seq(
```

```
  Row(7369,"SMITH","CLERK",800,20),
  Row(7499,"ALLEN","SALESMAN",1600,30),
  Row(7521,"WARD","SALESMAN",1250,30),
  Row(7566,"JONES","MANAGER",2975,20),
  Row(7654,"MARTIN","SALESMAN",1400,30),
  Row(7698,"BLAKE","MANAGER",2850,30)
)
var dfFromData1  = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData1.show()
```

```
+-----+------+--------+----+------+
|Empno| Ename|     Job| Sal|DeptId|
+-----+------+--------+----+------+
| 7369| SMITH|   CLERK| 800|    20|
| 7499| ALLEN|SALESMAN|1600|    30|
| 7521|  WARD|SALESMAN|1250|    30|
| 7566| JONES| MANAGER|2975|    20|
| 7654|MARTIN|SALESMAN|1400|    30|
| 7698| BLAKE| MANAGER|2850|    30|
+-----+------+--------+----+------+
```

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window

//row_number
val windowSpec   = Window.partitionBy("DeptId").orderBy("Sal")
dfFromData1.withColumn("rank",row_number.over(windowSpec)).show()
```

```
+-----+------+--------+----+------+----+
|Empno| Ename|     Job| Sal|DeptId|rank|
+-----+------+--------+----+------+----+
| 7369| SMITH|   CLERK| 800|    20|   1|
| 7566| JONES| MANAGER|2975|    20|   2|
| 7521|  WARD|SALESMAN|1250|    30|   1|
| 7654|MARTIN|SALESMAN|1400|    30|   2|
| 7499| ALLEN|SALESMAN|1600|    30|   3|
| 7698| BLAKE| MANAGER|2850|    30|   4|
+-----+------+--------+----+------+----+
```

Requirement 13:

```
Seq(
        ROW(7369,"SMITH","CLERK",800,20)
```

```
        ROW(7499,"ALLEN","SALESMAN",1600,30)
        ROW(7521,"WARD","SALESMAN",1250,30)
        ROW(7566,"JONES","MANAGER",2975,20)
        ROW(7654,"MARTIN","SALESMAN",1400,30)
        ROW(7698,"BLAKE",MANAGER",2850,30)
)
```

You need to write a Scala program using Spark to perform

1. Create a Data Frame from the above Sequence
2. Display the Emp name and his salary along with the previous employee's salary inside every department.
3. Display the Emp name and his salary along with the next employee's salary inside every department.

```
import org.apache.spark.sql.types.{StringType, StructField, StructType, ArrayType,IntegerType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._
val schema = StructType( Array(
          StructField("Empno", IntegerType,true),
          StructField("Ename",StringType,true),
          StructField("Job", StringType,true),
          StructField("Sal", IntegerType,true),
          StructField("DeptId", IntegerType,true),
        ))
val data =Seq(
  Row(7369,"SMITH","CLERK",800,20),
  Row(7499,"ALLEN","SALESMAN",1600,30),
  Row(7521,"WARD","SALESMAN",1250,30),
  Row(7566,"JONES","MANAGER",2975,20),
  Row(7654,"MARTIN","SALESMAN",1400,30),
  Row(7698,"BLAKE","MANAGER",2850,30)
)
var dfFromData1 = spark.createDataFrame(spark.sparkContext.parallelize(data),schema)
dfFromData1.show()
```

```
+-----+------+---------+----+------+
|Empno| Ename|      Job| Sal|DeptId|
+-----+------+---------+----+------+
| 7369| SMITH|    CLERK| 800|    20|
| 7499| ALLEN|SALESMAN|1600|    30|
| 7521|  WARD|SALESMAN|1250|    30|
| 7566| JONES|  MANAGER|2975|    20|
| 7654|MARTIN|SALESMAN|1400|    30|
| 7698| BLAKE|  MANAGER|2850|    30|
+-----+------+---------+----+------+
```

```
dfFromData1.createOrReplaceTempView("EmpDetails")
spark.sql("SELECT Ename,Sal,LAG(sal) OVER (PARTITION BY DeptId Order BY Sal) As PreviousSal FROM EmpDetails").show(false)
```

```
+------+----+------------+
|Ename |Sal |PreviousSal|
+------+----+------------+
|SMITH |800 |null        |
|JONES |2975|800         |
|WARD  |1250|null        |
|MARTIN|1400|1250        |
|ALLEN |1600|1400        |
|BLAKE |2850|1600        |
+------+----+------------+
```

```
spark.sql("SELECT Ename,Sal,LEAD(sal) OVER (PARTITION BY DeptId Order BY Sal) As NextSal FROM EmpDetails").show(false)
```

```
+------+----+-------+
|Ename |Sal |NextSal|
+------+----+-------+
|SMITH |800 |2975   |
|JONES |2975|null   |
|WARD  |1250|1400   |
|MARTIN|1400|1600   |
|ALLEN |1600|2850   |
|BLAKE |2850|null   |
+------+----+-------+
```

Requirement 14:

We have the emp data available in a Sequence as Follows

Seq(

ROW(7369,"SMITH","CLERK",800,20)
ROW(7499,"ALLEN","SALESMAN",1600,30)
ROW(7521,"WARD","SALESMAN",1250,30)
ROW(7566,"JONES","MANAGER",2975,20)
ROW(7654,"MARTIN","SALESMAN",1400,30)
ROW(7698,"BLAKE",MANAGER",2850,30)
)

We have a HBase table with the following structure

Key	Column Family : Person
empId	Ename, Job, Sal, Deptno

You need to write a Scala program using Spark to perform

1. Create a Data Frame from the above Sequence
2. Store the data from the Data From into the HBase table.

Catalog for HBase Table EMP

```
def catalog =
    s"""{
      |"table":{"namespace":"default", "name":"emp"},
      |"rowkey":"empId",
      |"columns":{
      |"key":{"cf":"rowkey", "col":"empId", "type":"int"},
      |"eName":{"cf":"person", "col":"eName", "type":"string"},
      |"job":{"cf":"person", "col":"job", "type":"string"},
      |"sal":{"cf":"person","col":"sal", "type":"string"},
      |"deptno":{"cf":"person", "col":"deptno", "type":"string"},
      |}
      |}""".stripMargin
```

Ans:

```
val df= Seq(
(7369,"SMITH","CLERK",800,20) ,
(7499,"ALLEN","SALESMAN",1600,30),
(7521,"WARD","SALESMAN",1250,30),
```

```
(7566,"JONES","MANAGER",2975,20),
(7654,"MARTIN","SALESMAN",1400,30),
(7698,"BLAKE",MANAGER",2850,30)
).toDF("ID","NAME","DEPT","SAL","LEAVES")

def catalog =
    s"""{
      |"table":{"namespace":"default", "name":"emp"},
      |"rowkey":"empId",
      |"columns":{
      |"key":{"cf":"rowkey", "col":"empId", "type":"int"},
      |"eName":{"cf":"person", "col":"eName", "type":"string"},
      |"job":{"cf":"person", "col":"job", "type":"string"},
      |"sal":{"cf":"person", "col":"sal", "type":"string"},
      |"deptno":{"cf":"person", "col":"deptno", "type":"string"},
      |}
      |}""".stripMargin

case class HBaseRecord(
  col0: Int,
  col1: String,
  col2: String,
  col3: Int,
  col4: Int)

object HBaseRecord
{
  def apply(i: Int, t: String): HBaseRecord = {
    val s = s"""row${"%03d".format(i)}"""
    HBaseRecord(i,
    s"String$i: $t",
    s"String$i: $t"
    i,
```

```
    i)
  }
}

val data = (0 to 255).map { i =>  HBaseRecord(i, "df")}

sc.parallelize(data).toDF.write.options(
 Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()

def withCatalog(cat: String): DataFrame = {
  sqlContext
  .read
  .options(Map(HBaseTableCatalog.tableCatalog->cat))
  .format("org.apache.hadoop.hbase.spark")
  .load()
}
val df = withCatalog(catalog)
```

Important Instructions:

- Please read the document thoroughly before you code.
- Please do not change the Business Requirements.

Coverage:

9. SQOOP
10. HBase
11. Cassandra
12. MongoDB
13. Scala

For all the hands-on, please consider the following,

UNIX User name: training

Home Directory: /home/training

SQOOP

Requirement 1:

We have the table named EMP which is inside the database named MyDB in the local instance of the MySQL. You are supposed to write a SQOOP command to read the data from the table and put inside a file with comma separated values in HDFS. The credentials are given below,

Server instance: localhost
Username: root
Password: password-1

Ans: sqoop-import –connect jdbc:mysql://localhost/mydb –username root –table EMP –target-dir /mydbemp -m 1 -P

```
hduser@hadoop:/usr/local/hadoop-2.6.0$ hdfs dfs -cat /mydbemp/part*
7369,SMITH,CLERK,7902,1980-12-17,800,null,20
7499,ALLEN,SALESMAN,7698,1981-02-20,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02,2975,null,20
7854,MARTIN,SALESMAN,7698,1981-09-28,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01,2850,null,30
7782,CLARK,MANAGER,7839,1981-06-06,2450,null,10
7788,SCOTT,ANALYST,7756,1987-04-19,3000,null,20
```

Requirement 2:

We have the table named EMP which is inside the database named MyDB in the local instance of the MySQL. You are supposed to write a SQOOP command to read the employees who work as MANAGER from the table and put inside a file with comma separated values inside the directory named MGR under HDFS. The credentials are given below,

Server instance: localhost
Username: root
Password: password-1

Ans: sqoop-import –connect jdbc:mysql://localhost/mydb –username root –table EMP –where "job='MANAGER'" –target-dir /empmgr -m 1 -P

```
21/05/28 14:09:21 INFO mapreduce.ImportJobBase: Transferred 144 bytes in 18.5591 seconds (7.759 bytes/sec)
21/05/28 14:09:21 INFO mapreduce.ImportJobBase: Retrieved 3 records.
hduser@hadoop:/usr/local/hadoop-2.6.0$ hdfs dfs -cat /empmgr/part*
7566,JONES,MANAGER,7839,1981-04-02,2975,null,20
7698,BLAKE,MANAGER,7839,1981-05-01,2850,null,30
7782,CLARK,MANAGER,7839,1981-06-06,2450,null,10
hduser@hadoop:/usr/local/hadoop-2.6.0$
```

Requirement 3:

We have the already imported the data from the table EMP into HDFS in the 1st requirement. Few new employees have been added and the newly added employees should be appended at the end of file located in HDFS. You are supposed to write a SQOOP command to read the newly added employee data from the table and append at the end of file in HDFS. The credentials are given below,

Server instance: localhost
Username: root

Password: password-1

Ans: sqoop-import –connect jdbc:mysql://localhost/mydb –username root –table EMP –incremental append –check-column empno –last-value 7788 –target-dir /mydbemp -m 1 -P

```
hduser@hadoop:/usr/local/hadoop-2.6.0$ hdfs dfs -cat /mydbemp/part*
7369,SMITH,CLERK,7902,1980-12-17,800,null,20
7499,ALLEN,SALESMAN,7698,1981-02-20,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02,2975,null,20
7854,MARTIN,SALESMAN,7698,1981-09-28,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01,2850,null,30
7782,CLARK,MANAGER,7839,1981-06-06,2450,null,10
7788,SCOTT,ANALYST,7756,1987-04-19,3000,null,20
7854,MARTIN,SALESMAN,7698,1981-09-28,1250,1400,30
7839,KING,PRESIDENT,null,1981-11-17,5000,null,10
7844,TURNER,SALESMAN,7698,1981-09-08,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-27,1100,null,20
hduser@hadoop:/usr/local/hadoop-2.6.0$
```

Requirement 4:

We have six tables available in the database named MyDB in MySQL instance. Requirement is load all the six tables' data into files (Each file should be placed in a separate directory) in HDFS. You are supposed to write a SQOOP command to perform this task. The credentials are given below,

Server instance: localhost

Username: root

Password: password-1

Ans: sqoop import-all-tables –connect jdbc:mysql://localhost/mydb –username root –warehouse-dir /mydball -m 1 -P

```
hduser@hadoop:/usr/local/hadoop-2.6.0$ hdfs dfs -ls /mydball
Found 3 items
drwxr-xr-x   - hduser supergroup          0 2021-05-28 14:37 /mydball/EMP
drwxr-xr-x   - hduser supergroup          0 2021-05-28 14:37 /mydball/customer
drwxr-xr-x   - hduser supergroup          0 2021-05-28 14:38 /mydball/product
hduser@hadoop:/usr/local/hadoop-2.6.0$
```

Requirement 5:

We have a file named Product located in HDFS. We have a new table named Product created in the database MyDB. Requirement is to export the data in the product file into the MySQL database table named Product. The credentials are given below,

Server instance: localhost
Username: root
Password: password-1

The product file and the script to create the product table is given below.


product.txt


Script_product.txt

```
        File Output Format Counters
                Bytes Written=0
21/05/28 12:28:09 INFO mapreduce.ExportJobBase: Transferred 280 bytes in 23.5529 seconds (11.8881 bytes/sec)
21/05/28 12:28:09 INFO mapreduce.ExportJobBase: Exported 10 records.
```

```
mysql> select * from product;
+------+----------+------+-------+
| slno | pname    | qty  | price |
+------+----------+------+-------+
|    1 | Shirt    |   10 |  1000 |
|    2 | Shoe     |   20 |  2000 |
|    3 | Trouser  |   10 |  2500 |
|    4 | Umbrella |   30 |   250 |
|    5 | Pen      |   30 |    50 |
|    6 | Pencil   |   30 |    10 |
|    7 | Cabbage  |   50 |    60 |
|    8 | Fish     |   40 |   100 |
|    9 | Carrot   |   50 |    50 |
|   10 | Beans    |   50 |    65 |
+------+----------+------+-------+
10 rows in set (0.01 sec)
```

Requirement 6:

We have the table named Product created in the database MyDB (Populated by the previous activity). Requirement is to read the data from the MySQL table Product and load the data in the Hive table. The credentials are given below,

Server instance: localhost
Username: root
Password: password-1

Write SQOOP commands to load the data from MySQL table to hive table, when

1. In hive, when there is no table with the name Product.

   Ans: sqoop-import –connect jdbc:mysql://localhost/mydb –username root –table product –hive-import -m 1 -P

2. In hive, we have the product table existing.

   Ans: sqoop-import –connect jdbc:mysql://localhost/mydb –username root –table product –hive-overwrite -m 1 -P

```
hive> select * from product;
OK
1       Shirt   10      1000
2       Shoe    20      2000
3       Trouser 10      2500
4       Umbrella        30      250
5       Pen     30      50
6       Pencil  30      10
7       Cabbage 50      60
8       Fish    40      100
9       Carrot  50      50
10      Beans   50      65
Time taken: 0.569 seconds, Fetched: 10 row(s)
```

HBase

Requirement 7:

We need to have an HBase table named 'users' as follows

Row Key	Column Family "Name"	Column Family "Image"
user1	Fname: Mahesh, Lname: Kumar	<mahesh.jpg>
user2	Fname: David, Lname: John	<david.jpg>
user3	Fname: Anand, Lname: Balaji	<anand.jpg>

Write commands,

i. To create the table

```
hbase(main):001:0> create 'users','Name','Image'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-0.98.17-hadoop2/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinde
r.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
0 row(s) in 4.4020 seconds

=> Hbase::Table - users
hbase(main):002:0> list
TABLE
empl
users
2 row(s) in 0.1080 seconds

=> ["empl", "users"]
```

ii. To insert the above 3 rows

```
hbase(main):011:0> put 'users','user3','Name:Fname','Anand'
0 row(s) in 0.0410 seconds

hbase(main):012:0> put 'users','user3','Name:Lname','Balaji'
0 row(s) in 0.0420 seconds

hbase(main):013:0> put 'users','user3','Image:image','<anand.jpg>'
0 row(s) in 0.0290 seconds

hbase(main):014:0> scan 'users'
ROW                                COLUMN+CELL
 user1                             column=Image:image, timestamp=1622131408162, value=<mahesh.jpg>
 user1                             column=Name:Fname, timestamp=1622131305480, value=Mahesh
 user1                             column=Name:Lname, timestamp=1622131336918, value=Kumar
 user2                             column=Image:image, timestamp=1622131503588, value=<david.jpg>
 user2                             column=Name:Fname, timestamp=1622131459351, value=David
 user2                             column=Name:Lname, timestamp=1622131482033, value=John
 user3                             column=Image:image, timestamp=1622131591383, value=<anand.jpg>
 user3                             column=Name:Fname, timestamp=1622131547479, value=Anand
 user3                             column=Name:Lname, timestamp=1622131568101, value=Balaji
3 row(s) in 0.0910 seconds
```

Requirement 8:

Assume that we have 10 rows in 'users' table. Write appropriate commands to perform the following

i. to read all rows from 'users' table

Ans: **scan 'users'**

ii. To read the first 3 rows from 'users' table

Ans: **scan 'users',{STOPROW=>'user4'}**

iii. To read only the column family "Name"

```
hbase(main):023:0> scan 'users',{COLUMNS=>['Name:Fname','Name:Lname']}
ROW                                   COLUMN+CELL
 user1                                column=Name:Fname, timestamp=1622134501788, value=Mahesh
 user1                                column=Name:Lname, timestamp=1622134529596, value=Kumar
 user2                                column=Name:Fname, timestamp=1622134596414, value=David
 user2                                column=Name:Lname, timestamp=1622134615999, value=John
 user3                                column=Name:Fname, timestamp=1622134666790, value=Anand
 user3                                column=Name:Lname, timestamp=1622134703261, value=Balaji
3 row(s) in 0.0460 seconds
```

iv. To read the rows starting from 4th rows up to the 10th row

Ans: **scan 'users',{STARTROW=>'user4'}**

Requirement 9:

Assume that we have 10 rows in 'users' table. Write appropriate commands to perform the following

i. to delete the Lname value for the 2nd row

```
hbase(main):024:0> delete 'users','user2','Name:Lname',1622134615999
0 row(s) in 0.0830 seconds

hbase(main):025:0> scan 'users'
ROW                                   COLUMN+CELL
 user1                                column=Image:image, timestamp=1622134562135, value=<mahesh.jpg>
 user1                                column=Name:Fname, timestamp=1622134501788, value=Mahesh
 user1                                column=Name:Lname, timestamp=1622134529596, value=Kumar
 user2                                column=Image:image, timestamp=1622134636595, value=<david.jpg>
 user2                                column=Name:Fname, timestamp=1622134596414, value=David
 user3                                column=Image:image, timestamp=1622134719499, value=<anand.jpg>
 user3                                column=Name:Fname, timestamp=1622134666790, value=Anand
 user3                                column=Name:Lname, timestamp=1622134703261, value=Balaji
3 row(s) in 0.0940 seconds
```

ii. to remove 3rd row from 'users' table

```
hbase(main):026:0> deleteall 'users','user3'
0 row(s) in 0.0450 seconds

hbase(main):027:0> scan 'users'
ROW                                COLUMN+CELL
 user1                             column=Image:image, timestamp=1622134562135, value=<mahesh.jpg>
 user1                             column=Name:Fname, timestamp=1622134501788, value=Mahesh
 user1                             column=Name:Lname, timestamp=1622134529596, value=Kumar
 user2                             column=Image:image, timestamp=1622134636595, value=<david.jpg>
 user2                             column=Name:Fname, timestamp=1622134596414, value=David
2 row(s) in 0.0380 seconds
```

iii. to remove all the rows from the table

```
hbase(main):028:0> truncate 'users'
Truncating 'users' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 2.1630 seconds

hbase(main):029:0> scan 'users'
ROW                                COLUMN+CELL
0 row(s) in 0.3910 seconds
```

Requirement 10:

We have an HBase table named 'users' as follows

Row Key	**Column Family "Name"**	**Column Family "Image"**
user1	Fname: Mahesh, Lname: Kumar	<mahesh.jpg>
user2	Fname: David, Lname: John	<david.jpg>
user3	Fname: Anand, Lname: Balaji	<anand.jpg>

Write commands,

i. To add a new Column family named 'Email'

```
hbase(main):032:0> alter 'users','Email'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.7980 seconds

hbase(main):033:0> describe 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'Email', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE'
, TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'Image', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE'
, TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'Name', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
 TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
3 row(s) in 0.2000 seconds
```

ii. To remove the existing Column family named 'Email'

```
hbase(main):034:0> alter 'users','delete'=>'Email'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.9430 seconds

hbase(main):035:0> describe 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'Image', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE'
, TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'Name', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
 TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.1580 seconds
```

Requirement 11:

We have the following movies.dat file available under data folder in HDFS.

movies.dat

Create an HBase table named movies and load the contents of the above file into movies table.

Ans: hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns="HBASE_ROW_KEY,mn:name,mn:id,mn:duration" -Dimporttsv.separator="," movie /inputdir/data/movies.dat

Requirement 12:

We need to have an HBase table named 'users' as follows

Row Key	Column Family "Name"	Column Family "Image"
user1	Fname: Mahesh, Lname: Kumar	<mahesh.jpg>
user2	Fname: David, Lname: John	<david.jpg>
user3	Fname: Anand, Lname: Balaji	<anand.jpg>

To create the table using **HBaseAdmin.**

```
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.conf.Configuration;

public class CreateTable {

  public static void main(String[] args) throws IOException {

    // Instantiating configuration class
    Configuration con = HBaseConfiguration.create();

    // Instantiating HbaseAdmin class
    HBaseAdmin admin = new HBaseAdmin(con);

    // Instantiating table descriptor class
    HTableDescriptor tableDescriptor = new
    HTableDescriptor(TableName.valueOf("emp"));

    // Adding column families to table descriptor
    tableDescriptor.addFamily(new HColumnDescriptor("personal"));
    tableDescriptor.addFamily(new HColumnDescriptor("professional"));
```

```
    // Execute the table through admin
    admin.createTable(tableDescriptor);
    System.out.println(" Table created ");
  }
}
```

Cassandra

Requirement 13:

In Cassandra, Create a key space named **DEMO** and create a table called student with the following columns

regno INT
sname TEXT
mark1 INT
mark2 INT
mark3 INT

Add the following data in the student table,
1000,'RAJ',56,76,91
1001,'AMIT',96,86,91
1002,'DINESH',82,81,84

```
cqlsh> CREATE KEYSPACE DEMO WITH REPLICATION = {'class':'SimpleStrategy','replication_factor':3};
cqlsh> describe keyspaces

demo           system_auth  vivaz               system_traces
system_schema  system       system_distributed
```

```
cqlsh:demo> create table student(regno int primary key,sname text,mark1 int,mark2 int,mark3 int);
cqlsh:demo> describe student;

CREATE TABLE demo.student (
    regno int PRIMARY KEY,
    mark1 int,
    mark2 int,
    mark3 int,
    sname text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

```
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1000,'RAJ',56,76,91);
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1001,'AMIT',96,86,91);
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1002,'DINESH',82,81,84);
cqlsh:demo> select * from student;

 regno | mark1 | mark2 | mark3 | sname
-------+-------+-------+-------+--------
  1001 |    96 |    86 |    91 |   AMIT
  1002 |    82 |    81 |    84 | DINESH
  1000 |    56 |    76 |    91 |    RAJ

(3 rows)
```

Write a Query to display the details of the students whose name is AMIT.


```
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1000,'RAJ',56,76,91);
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1001,'AMIT',96,86,91);
cqlsh:demo> insert into student(regno,sname,mark1,mark2,mark3) values(1002,'DINESH',82,81,84);
cqlsh:demo> select * from student;

 regno | mark1 | mark2 | mark3 | sname
-------+-------+-------+-------+--------
  1001 |    96 |    86 |    91 |   AMIT
  1002 |    82 |    81 |    84 | DINESH
  1000 |    56 |    76 |    91 |    RAJ

(3 rows)
```


MONGO DB

Requirement 14:

A website which provides Online Trainings on various technologies, wants to record all the enquires in MongoDB. You need to create a document to achieve this, based on the following requirements

1. Course Name
2. Duration and Trainer of the course

3. Batch type – Fast Track, Regular or Weekend
4. Category.

```
> show databases;
admin   0.000GB
config  0.000GB
local   0.000GB
vivaz   0.000GB
> use mydb;
switched to db mydb
> show databases;
admin   0.000GB
config  0.000GB
local   0.000GB
vivaz   0.000GB
> db.createCollection('Online_training')
{ "ok" : 1 }
> db.Online_training.insert({
... Course_name: "BigData",
... Duration: "6 months",
... taken_by:"Karthick",
... Batch_Type:["Fast_Track","Regular","Weekend"],
... Category: "DW04"
... })
WriteResult({ "nInserted" : 1 })
```

```
> use mydb;
switched to db mydb
> db.Online_training.find().pretty()
{
        "_id" : ObjectId("60b5e4e9a156beec3d292e69"),
        "Course_name" : "BigData",
        "Duration" : "6 months",
        "taken_by" : "Karthick",
        "Batch_Type" : [
                "Fast_Track",
                "Regular",
                "Weekend"
        ],
        "Category" : "DW04"
}
```

SCALA

Requirement 15:

Write a Scala program that includes a class named Rational to generate the rational with the following requirement

Re-implement the toString method

Check the divisor part should not be zero

Example:

When the object is constructed as

val r = new Rational(1, 2)

The output should be,

r: Rational = 1/2

When the object is constructed as

val r = new Rational(66, 42)

The output should be,

r: Rational = 11/7

Ans:

```
class Rational(n: Int, d: Int) {
   require(d != 0)
   private val g = gcd(n.abs, d.abs)
   val numer = n / g
   val denom = d / g
   override def toString = "Rational = "+numer+"/"+ denom
   private def gcd(a: Int, b: Int): Int =
     if (b == 0) a else gcd(b, a % b)
 }
object HelloWorld {
  def main(args: Array[String]) {
     val r = new Rational(12, 8)
    println(r)
  }
}
```

Requirement 16:

Modify the above Scala program to include the overloading of operators as follows,

When the object is constructed as

val r = new Rational(2, 3)

The output should be,

r: Rational = 2/3

Now, when you say

r * r

The output should be,

r: Rational = 4/9

And when you say

r * 2

The output should be,

r: Rational = 4/3

So all the operators (+, -, *, /) should be overloaded to either take a Rational or a number next to it.

Ans:

```
class Rational(n: Int, d: Int) {
   require(d != 0)
   private val g = gcd(n.abs, d.abs)
   val numer = n / g
   val denom = d / g
   def this(n: Int) = this(n, 1)
   def + (that: Rational): Rational =
     new Rational(
       numer * that.denom + that.numer * denom,
       denom * that.denom
     )
   def + (i: Int): Rational =
     new Rational(numer + i * denom, denom)
   def - (that: Rational): Rational =
     new Rational(
       numer * that.denom - that.numer * denom,
       denom * that.denom
     )
   def - (i: Int): Rational =
     new Rational(numer - i * denom, denom)
   def * (that: Rational): Rational =
     new Rational(numer * that.numer, denom * that.denom)
   def * (i: Int): Rational =
     new Rational(numer * i, denom)
   def / (that: Rational): Rational =
     new Rational(numer * that.denom, denom * that.numer)
   def / (i: Int): Rational =
     new Rational(numer, denom * i)
   override def toString = "Rational = "+numer+"/"+ denom
private def gcd(a: Int, b: Int): Int =
     if (b == 0) a else gcd(b, a % b)
  }
object HelloWorld {
```

```
  def main(args: Array[String]) {
     val r = new Rational(12, 8)
      println(r*r)
  }
}
```

Requirement 17:

Write a function in Scala to return the bigger of 2 numbers passed as parameters with the following signature

max: (Int,Int)Int

Ans: object greater{

```
 def main(args:Array[String]){
  println(max(2,5)+" is greater");
 }
 def max(a:Int,b:Int):Int={
  if(a>b){
   return a;
  }
  else
   return b;
 }
}
```

```
5 is greater
```

Requirement 18:

Write a Scala program that accepts a set of numbers as Command line parameters and display the sum of all the numbers

For Example, when this program is executed in the Command prompt as,

Scala sum.scala 100 25 200 50

The output should be,

375

Ans:object sumarg{

```
 def main(args:Array[String]){
  var sum=0;
  for(arg<-args){
     sum+=arg.toInt;
  }
  println(sum)
 }
}
```

```
375
```

Requirement 19:

Write a Scala program and introduce 2 list lists with few values. Introduce a tuple named employee with ename and salary. Introduce a set named fruit with the values Apple and Orange. Introduce a map named TaskMap and with 2 tasks - 1. Read Big Data and 2. Understand Map Reduce. Write code to perform the following

1. Combine both the list and put them into the 3rd list and display the 3rd list
2. Display each of the value of the employee tuple
3. Add one more fruit Banana into the set and print the set
4. Add one more task 3.Try HDFS Commands and then print the 2nd task in the Map.

Ans:

```
import scala.collection.mutable.Set
import scala.collection.mutable.Map
object testCollections{
 def main(args:Array[String]){
  val list1 = List(1,2,3);
  val list2 = List(4,5,6);
  val list3 = list1:::list2;
  println("List3 is: "+list3);
  val employee = ("Ramesh",30000,"Ganesh",50000);
  employee.productIterator.foreach{i=>println(i)};
  var fruit = Set("Apple","Orange")
  fruit+="Banana"
```

```
  println(fruit)
  var TaskMap = Map(1->"Read Big Data and", 2->"Understand Map Reduce");
  TaskMap+=(3->"Try HDFS Commands")
  println(TaskMap(2))
 }
}
```

```
List3 is: List(1, 2, 3, 4, 5, 6)
Ramesh
30000
Ganesh
50000
HashSet(Apple, Orange, Banana)
Understand Map Reduce

warning: there was one deprecation warning (since 2.13.
```

Requirement 20:

Write a Scala program to find the GCD (Greatest Common Divisor) of 2 numbers. For example, if we consider the following 2 numbers 42 and 14, the GCD should be 7.

Ans:

```
object findGCD{
 def main(args:Array[String]){
  var a=63;
  var b =42;
  var r = b;
  while(a%b!=0){
     r=a%b;
     a=b;
     b=r;
  }
  println("GCD of 42 and 63= "+r)
 }
}
```

```
GCD of 42 and 63= 21

warning: there was one deprecation warning (since 2.13.0); re-run with -deprecation for details
```

Requirement 21:

Write a Scala program to introduce a String variable and reverse the string. For Example, if the string is initialized to 'Welcome',

The output should be 'emocleW'

Ans:

```
object revString {
  def reverseString(newString: String): String = {
    var revString = ""
    val n = newString.length()
    for(i <- 0 to n-1){
      revString = revString.concat(newString.charAt(n-i-1).toString)
    }
  return revString
  }
  def main(args: Array[String]) {
    var newString = "Welcome"
    println("Reverse of '" + newString + "' is '" + reverseString(newString) + "'")
  }
}
```

```
Reverse of 'Welcome' is 'emocleW'

warning: there was one deprecation warning (since 2.13.0); r
```

Requirement 22:

Write a Scala program to list the Scala program files in the current directory (File names ends with '.scala')

Requirement 23:

Write a Scala function that takes Amount as parameter to find the minimum number of denominations.
For Example, if the amount is 5000, then

2000 * 2 = 4000

500 * 2 = 1000

So, minimum no of denomination for 5000 = 4

For Example, if the amount is 8941, then

2000 * 4 = 8000

500 * 1 = 500

200 * 2 = 400

20 * 2 = 40

1 * 1 = 1

So, minimum no of denomination for 8941 = 10

Ans:

```
object Demo {
  var z = Array(2000, 1000, 500, 200, 100, 50, 20, 10, 1)
  var count: Int = 0
  def denomination(amount: Int) {
    var temp: Int = amount
    var den: Int = 0
    for (i <- 0 to z.length - 1) {
      count = temp / z(i)
      if (count >= 1) {
        println(z(i) + " x " + count + " = " + z(i) * count);
        temp = temp - z(i) * count
        den = den + count
      }
    }
    print("The minimum number of denomination for "+amount+" ="+den);
  }
  def main(args: Array[String]) {
    denomination(8941)
```

```
 }
}
```

Requirement 24:

Write a Scala function that takes the number of rows as parameter and generate the Pyramid of numbers for the specified rows.
For Example, if the number of rows is 5, then

```
                1
            1   2   1
        1   2   3   2   1
    1   2   3   4   3   2   1
1   2   3   4   5   4   3   2   1
```

Requirement 25:

Write a Scala program that takes the filename, include a function to display the longest line in the file. The line number, line and the length of the line should be displayed as follows

LineNo	Line	length
…	…	…

Requirement 26:

Write a Scala program to introduce an array of numbers. Implement Bubble Sort to arrange the elements of the array in ascending order. Print the sorted array.

Requirement 27:

Create a list that has 100 values from 1 to 100. From the list, take all the even numbers and load them into the 2nd List. Except the numbers lies between 20 to 50 from the 2nd list, load the rest of the numbers into the 3rd List. Create the 4th list that should has double the values in 3rd list.

Requirement 28:

Create a Set with 3 elements – 1, 2 and 3. Add a list that has (10, 20, 30). Now add a value 5 to the set and then add 2 more values 6 and 7 to the set. Apply your logic to ensure the Set has only the values greater than 2. If the set has a value 10 then remove the value from the List.