**Word count in pig:**

input = load 'mary' as (line);

words = foreach input generate flatten(TOKENIZE(line)) as word;

grpd = group words by word;

cntd = foreach grpd generate group, COUNT(words);

dump cntd;

**Max temperature in pig:**

records = LOAD 'sample.txt'

AS (year:chararray, temperature:int, quality:int);

filtered\_records = FILTER records BY temperature != 9999 AND

(quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);

grouped\_records = GROUP filtered\_records BY year;

max\_temp = FOREACH grouped\_records GENERATE group, MAX(filtered\_records.temperature);

DUMP max\_temp;

**Number of Products Sold in Each Country in pig:**

salesTable = LOAD '/SalesJan2009.csv' USING PigStorage(',') AS

(Transaction\_date:chararray,Product:chararray,Price:chararray,Payment\_Type:chararray,Name:chararray,

City:chararray,State:chararray,Country:chararray,Account\_Created:chararray,Last\_Login:chararray,Latitude:chararray,Longitude:chararray);

GroupByCountry = GROUP salesTable BY Country;

CountByCountry = FOREACH GroupByCountry GENERATE CONCAT((chararray)$0,CONCAT(':',(chararray)COUNT($1)));

STORE CountByCountry INTO 'pig\_output\_sales' USING PigStorage('\t');

**UDF pig:**

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import org.apache.pig.FilterFunc;

import org.apache.pig.backend.executionengine.ExecException;

import org.apache.pig.data.DataType;

import org.apache.pig.data.Tuple;

import org.apache.pig.impl.logicalLayer.FrontendException;

public class IsGoodQuality extends FilterFunc {

@Override

public Boolean exec(Tuple tuple) throws IOException {

if (tuple == null || tuple.size() == 0) {

return false;

}

try {

Object object = tuple.get(0);

if (object == null) {

return false;

}

int i = (Integer) object;

return i == 0 || i == 1 || i == 4 || i == 5 || i == 9;

} catch (ExecException e) {

throw new IOException(e);

}

}

}

**Execution inTerminal:**

REGISTER pig-examples.jar;

DEFINE isGood com.hadoopbook.pig.IsGoodQuality();

records = LOAD 'sample.txt' AS (year:chararray, temperature:int, quality:int);

filtered\_records = FILTER records BY temperature != 9999 AND isGood(quality);

grouped\_records = GROUP filtered\_records BY year;

max\_temp = FOREACH grouped\_records GENERATE group, MAX(filtered\_records.temperature);

DUMP max\_temp;

**Temperature in Hive:**

CREATE DATABASE custom;

CREATE TABLE temperature\_data

(

full\_date STRING,

zip INT,

temperature INT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH '/home/hduser073/dataset\_Session 14.txt'

INTO TABLE custom.temperature\_data;

**Query:**

Select \* from temperature\_data;

**Book recommendation in hive:**

CREATE TABLE users1(

user\_id int,

location String,

age int

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

load data local inpath '/home/hduser073/Desktop/users.csv' overwrite into table users1;

create table rating(

user\_id int,

isbn string,

book\_rating int

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

load data local inpath '/home/hduser073/Desktop/books.csv' overwrite into table books;

create table books(

isbn string,

title string,

author string,

yop int,

publisher string,

img\_url\_s string,

img\_url\_m string,

img\_url\_l string

)

row format delimited

fields terminated by ','

stored as textfile;

load data local inpath '/home/hduser073/Desktop/ratings.csv' overwrite into table rating;

**Queries:**

CREATE INDEX user\_index on TABLE users1 (user\_id) AS'COMPACT' WITH DEFERRED REBUILD;

Show index on users1;

CREATE INDEX user\_index on TABLE books (isbn) AS'COMPACT' WITH DEFERRED REBUILD;

Show index on books;

alter table rating set tblproperties("skip.head.line.count"="1");

alter table users1 set tblproperties("skip.head.line.count"="1");

alter table books set tblproperties("skip.head.line.count"="1");

1. How many users are given rating for any book?

select count(distinct user\_id) from rating;

2. Top 10 rated book?

select b.title,r.user\_id,r.book\_rating from books b join rating r on (b.isbn=r.isbn)

group by b.title,r.user\_id,r.book\_rating order by r.book\_rating desc limit 10;

3. How many books have not rated?

select count(b.isbn) from books b where b.isbn not in (select distinct isbn from rating);

4. Top 10 users who have given maximum rating for maximum book?

select isbn,count(isbn) as cnt from rating group by isbn order by cnt desc limit 10;

5. How many books rated by user id 104,200, and 700?

select count(user\_id) from rating where user\_id="104" or user\_id="200" or user\_id="700"

group by user\_id;

**Aadhar in hive:**

create table aad(

register string,

enrolment\_agency string,

state string,

district string,

sub\_district string,

pincode int,

gender string,

age int,

aad\_generated int,

enroll\_rejected int,

res\_providing\_email int,

res\_providing\_number int

)

row format delimited

fields terminated by ','

stored as textfile;

load data local inpath '/home/hduser073/Desktop/adata.txt' overwrite into table aad;

Queries:

1. Count the number of Identities generated in each state

select state,count(\*) from aad group by state;

2. Count the number of Identities generated by each Enrollment Agency

select enrolment\_agency,count(\*) from aad group by enrolment\_agency;

3. Top 10 districts with maximum identities generated for both Male and Female

select district,count(case when gender='M' then 1 end) as male\_cnt,count(case when gender='F' then 1 end) as female\_cnt from aad group by district order by male\_cnt desc,female\_cnt desc limit 10;

4. For how many states the Aadhaar data exists?

select count(distinct state) from aad;

5. What is the % of Aadhaar's being approved per state?

select state,((sum(aad\_generated)-sum(enroll\_rejected))/(sum(aad\_generated))\*100) from aad group by state;

**word count scala**

**==========**

val text = sc.textFile("mytextfile.txt")

val splitdata = text.flatMap(line => line.split(" "));

val mapdata = splitdata.map(word => (word,1));

val reducedata = mapdata.reduceByKey(\_+\_);

reducedata.collect;

**char count in**

**==================**

hdfs dfs -mkdir /spark

hdfs dfs -put /home/kgr/sparkdata.txt /spark

scala> val data=sc.textFile("sparkdata.txt");

val splitdata = data.flatMap(line => line.split(""));

val mapdata = splitdata.map(word => (word,1));

val reducedata = mapdata.reduceByKey(\_+\_);

reducedata.collect;

**Find the number of medals that India won year wise**

**==================**

val textFile = sc.textFile("hdfs://localhost:9000/olympix\_data.csv")

val counts = textFile.filter { x => {if(x.toString().split(",").length >= 10) true else false} }.map(line=>{line.toString().split(",")})

val fil = counts.filter(x=>{if(x(2).equalsIgnoreCase("india")&&(x(9).matches(("\\d+")))) true else false })

val pairs: RDD[(String, Int)] = fil.map(x => (x(3),x(9).toInt))

val cnt = pairs.reduceByKey(\_ + \_).collect()

**Transformations:**

**Filter:**

val x = sc.parallelize(Array(1,2,3))

val y = x.filter(n => n%2 == 1)

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))

**Flat map:**

val x = sc.parallelize(Array(1,2,3))

val y = x.flatMap(n => Array(n, n\*100, 42))

println(x.collect().mkString(", "))

println(y.colle**ct().mkString(", "))**

Group By:

val x = sc.parallelize( Array("John", "Fred", "Anna", "James"))

val y = x.groupBy(w => w.charAt(0))

println(y.collect().mkString(", "))

**Group By Key:**

val x = sc.parallelize( Array(('B',5),('B',4),('A',3),('A',2),('A',1)))

val y = x.groupByKey()

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))

**Zip:**

val x = sc.parallelize(Array(1,2,3))

val y = x.map(n=>n\*n)

val z = x.zip(y)

println(z.collect().mkString(", "))

**Actions:**

**Collect:**

val x = sc.parallelize(Array(1,2,3), 2)

val y = x.collect()

val xOut = x.glom().collect()

println(y)

**Sum:**

val x = sc.parallelize(Array(2,4,1))

val y = x.sum

println(x.collect().mkString(", "))

println(y)

**Max:**

val x = sc.parallelize(Array(2,4,1))

val y = x.max

println(x.collect().mkString(", "))

println(y)