Assignment 7.1

TASK1 - Write a program to implement wordcount using Pig.

SOLUTION#1

Here is the file on the local file system where we want to count number of words in. We will be using PIG in local mode for this purpose, though same script will work for HDFS case as well. Map Reduce output for each command has been removed here for clarity purposes and only PIG output is shown here.

```
                          • MobaXterm 10.4 •
            (SSH client, X-server and networking tools)

→ SSH session to acadgild@192.168.56.2
   • SSH compression : v
   • SSH-browser     : v
   • X11-forwarding  : v  (remote display is forwarded through SSH)
   • DISPLAY         : v  (automatically set on remote server)

→ For more info, ctrl+click on help or visit our website
```

```
Last login: Fri Jul 13 12:06:37 2018 from 192.168.56.1

#CHECK the presence of an input file for which we will run wordcount script
[acadgild.mmisra ~]$
[acadgild.mmisra ~]$ ls -la test.txt
-rw-rw-r--. 1 acadgild acadgild 336 Jun 17 11:14 test.txt
[acadgild.mmisra ~]$ cat test.txt
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
have a nice day
```

 # There are 21 rows of the same line in the file so we should get each word count as 21 when we run the PIG script.

```
[acadgild.mmisra ~]$
```

 # Launch PIG in local mode via -x local option

```
[acadgild.mmisra ~]$ pig -x local
```

 # Load each line of the file as a tuple(Line) in the lines bag. For that we use '\n' as the delimiter in PigStorage

```
grunt> lines = load 'test.txt' using PigStorage('\n') as (line:CharArray);
grunt>
```

#Check the content of the lines bag

```
grunt> dump lines;

(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
(have a nice day)
grunt>
grunt>
grunt>
```

 # Now we split each line into separate words using TOKENIZE method. Each word becomes a tuple in the words bag

```
grunt> words = foreach lines generate TOKENIZE(line);
grunt>
grunt>

grunt>
```

 # Check the contents of the words bag

```
grunt> dump words;

({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
```

```
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
({(have),(a),(nice),(day)})
grunt>

# Since each word is a separate tuple, we can flatten it into a single bag to contain
all the words. This will simplify the counting process

grunt>
grunt> flatwords = foreach words generate FLATTEN(bag_of_tokenTuples_from_line);
grunt>

# Check the contents after flattening
grunt> dump flatwords;
 (have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
```

```
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
(have)
(a)
(nice)
(day)
grunt>
```

 # Since we now have all the words in a single bag, we can group them. We are grouping them based on themselves which is the first and only field $0

```
grunt> groupwords = GROUP flatwords BY $0;
grunt>
grunt>
grunt>
```

 # Check the contents of the groupwords bag

```
grunt> dump groupwords;

(a,{(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a),(a
```

```
)})
(day,{(day),(day),(day),(day),(day),(day),(day),(day),(day),(day),(day),(day),(d
ay),(day),(day),(day),(day),(day),(day)})
(have,{(have),(have),(have),(have),(have),(have),(have),(have),(have),(have),(h
ave),(have),(have),(have),(have),(have),(have),(have),(have)})
(nice,{(nice),(nice),(nice),(nice),(nice),(nice),(nice),(nice),(nice),(nice),(n
ice),(nice),(nice),(nice),(nice),(nice),(nice),(nice),(nice)})
```

 # Now we can count each word using the COUNT statement for each tuple

```
grunt> wordcount = foreach groupwords generate (group),COUNT(flatwords);
grunt>
```

#Get the Real output by dumping wordcount

```
grunt> dump wordcount;
 (a,21)
(day,21)
(have,21)
(nice,21)
grunt>
grunt>
```

SUMMARY: Following commands were used for wordcount example using PIG

```
lines = load 'test.txt' using PigStorage('\n') as (line:CharArray);

dump lines;

words = foreach lines generate TOKENIZE(line);

dump words;

flatwords = foreach words generate FLATTEN(bag_of_tokenTuples_from_line);

dump flatworlds;

groupwords = GROUP flatwords BY $0;

dump groupwords;

wordcount = foreach groupwords generate (group),COUNT(flatwords);

dump wordcount;
```

TASK2 (SOLUTION)

We have employee_details and employee_expenses files. Use local mode while running Pig and
write Pig Latin script to get below results:

employee_details (EmpID,Name,Salary,DepartmentID)
https://github.com/prateekATacadgild/DatasetsForCognizant/blob/master/employee_details.txt
employee_expenses(EmpID,Expence)
https://github.com/prateekATacadgild/DatasetsForCognizant/blob/master/employee_expenses.txt

Launch PIG in local mode

```
[acadgild.mmisra ~]$ pig -x local
```

 # Read employee details and expenses in relations called details and expenses respectively

```
grunt> details = load 'employee_details.txt' using PigStorage(',') as
(EmpId:Int,Name:CharArray,Salary:Int,Dept:Int);

grunt> expenses = load 'employee_expenses.txt' using PigStorage(' ') as (EmpId:Int,Expense:Int);
```

(a) Top 5 employees (employee id and employee name) with highest rating. (In case two
employees have same rating, employee with name coming first in dictionary should get
preference)

 # SORT the details with the employee salary in descending order and limit the result to only 5

```
grunt> y = LIMIT sorted 5;
grunt>
grunt> dump y;
```

#here we get the answer
(106,Aamir,25000,1)
(101,Amitabh,20000,1)
(107,Salman,17500,2)
(108,Ranbir,14000,3)
(103,Akshay,11000,3)

(b) Top 3 employees (employee id and employee name) with highest salary, whose employee id
is an odd number. (In case two employees have same salary, employee with name coming first
in dictionary should get preference)

grunt>

 # Filter the sorted result for odd Employee ID and LIMIT the result to 3 records

```
grunt> filter_sorted= FILTER sorted BY ((EmpId%2) != 0);

grunt> ans2 =LIMIT filter_sorted 3;

grunt> dump ans2;
```

#here we get the answer

(101,Amitabh,20000,1)

(107,Salman,17500,2)

(103,Akshay,11000,3)

(c) Employee (employee id and employee name) with maximum expense (In case two
employees have same expense, employee with name coming first in dictionary should get
preference)

#Since there are many expenses with the same Employee ID, we will group the records based on the Employee ID first

```
grunt> expenses = load 'employee_expenses.txt' using PigStorage(' ') as
>> (EmpId:Int,Expense:Int);
2018-07-14 22:09:07,351 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-07-14 22:09:07,351 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
grunt> ex_g = group expenses BY EmpId;
grunt> dump ex_g;

(101,{(101,100),(101,200)})
(102,{(102,400),(102,100)})
(104,{(104,300)})
(105,{(105,100)})
```

(110,{(110,400)})
(114,{(114,200)})
(119,{(119,200)})
grunt>

 # then we calculate total expense by each employee ID by doing the sum for each record against the employee ID

```
grunt> ex_sum= foreach ex_g GENERATE group as EmpId,SUM(expenses.Expense) as sum;
grunt> dump ex_sum;
```

(101,300)
(102,500)
(104,300)
(105,100)
(110,400)
(114,200)
(119,200)

#now we know total expenses against each Employee ID. To find the MAX expense, we group the sums into another group

```
grunt> ex_m = group  ex_sum all;
grunt> dump ex_m;
```

(all,{(119,200),(114,200),(110,400),(105,100),(104,300),(102,500),(101,300)})

#Now we can find the max expense from this

```
grunt> ex_max = foreach ex_m generate group,MAX(ex_sum.sum) as max;
grunt> dump ex_max;
```

(all,500)

 # To know the Employee ID for the max expense, we filter the ex_sum relation where the expense is equal to the max(500)

```
grunt> ans = filter ex_sum BY sum==ex_max.max;
grunt> dump ans;
```

(102,500)

#To print the name and employee ID and the max expense, we join the details relation with the ans which will gives us the details of the EmpId having maximum expense

```
grunt> ans1 = JOIN details By EmpId,ans By EmpId;
grunt> dump ans1;
```

(102,Shahrukh,10000,2,102,500)

 # To print only the name, employee ID and expense we generate a new relation with only required fields

```
grunt> ans2 = foreach ans1 generate $0,$1,ans.sum;
```

```
grunt> dump ans2;
```

#Here we get the answer
(102,Shahrukh,500)

(d) List of employees (employee id and employee name) having entries in employee_expenses file.

 # We load the employee details from file in details relation

```
grunt> details = load 'employee_details.txt' using PigStorage(',') as
(EmpId:Int,Name:CharArray,Salary:Int,Dept:Int);
2018-07-14 22:24:39,778 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-07-14 22:24:39,778 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
```

 # We load the expenses from file in expense relation

```
expenses = load 'employee_expenses.txt' using PigStorage(' ') as
(EmpId:Int,Expense:Int);
```

 # We group the expenses based on the EmpId as there are multiple entries for the same EmpId

```
grunt> ex_g = group expenses BY EmpId;
```

 # the we sum the expenses of each Employee ID

```
grunt> ex_sum= foreach ex_g GENERATE group as EmpId,SUM(expenses.Expense) as sum;
grunt>
grunt>
grunt> dump ex_sum;
(101,300)
(102,500)
(104,300)
(105,100)
(110,400)
(114,200)
(119,200)
```

 # Now we can do inner join of details with ex_sum to list all the entries matching with EmpId

```
grunt> ans3 = JOIN details BY EmpId, ex_sum BY EmpId;
grunt> dump ans3;
```

(101,Amitabh,20000,1,101,300)
(102,Shahrukh,10000,2,102,500)
(104,Anubhav,5000,4,104,300)
(105,Pawan,2500,5,105,100)
(110,Priyanka,2000,5,110,400)
(114,Madhuri,2000,2,114,200)

 # Since we are interested in printing only EmpId and Name for people having entries in expenses, we generate a new relation with required fields

```
grunt> ans4 = foreach ans3 GENERATE $0,$1;
grunt>
grunt> dump ans4;
```

#Here we get the answer
(101,Amitabh)
(102,Shahrukh)
(104,Anubhav)
(105,Pawan)
(110,Priyanka)
(114,Madhuri)

(e) List of employees (employee id and employee name) having no entry in employee_expenses file.

 # For this we do Left outer join of details with the ex_sum by EmpId

```
grunt> ans5 = JOIN details BY EmpId LEFT OUTER, ex_sum BY EmpId;
grunt> dump ans5;
(101,Amitabh,20000,1,101,300)
(102,Shahrukh,10000,2,102,500)
(103,Akshay,11000,3,,)
(104,Anubhav,5000,4,104,300)
(105,Pawan,2500,5,105,100)
(106,Aamir,25000,1,,)
(107,Salman,17500,2,,)
(108,Ranbir,14000,3,,)
(109,Katrina,1000,4,,)
(110,Priyanka,2000,5,110,400)
(111,Tushar,500,1,,)
(112,Ajay,5000,2,,)
(113,Jubeen,1000,1,,)
(114,Madhuri,2000,2,114,200)
```

 # now we can get rid of the duplicate column for EmpID which is created after the join

grunt> ans6 = foreach ans5 generate ..$3,$5;

grunt> dump ans6;

(101,Amitabh,20000,1,300)

(102,Shahrukh,10000,2,500)

(103,Akshay,11000,3,)

(104,Anubhav,5000,4,300)

(105,Pawan,2500,5,100)

(106,Aamir,25000,1,)

(107,Salman,17500,2,)

(108,Ranbir,14000,3,)

(109,Katrina,1000,4,)

(110,Priyanka,2000,5,400)

(111,Tushar,500,1,)

(112,Ajay,5000,2,)

(113,Jubeen,1000,1,)

(114,Madhuri,2000,2,200)

#Then we filter only those records where the expense is NULL. As there are the employee who did not have any entry in the expenses file. We also print only required fields like EmpID and Name

grunt> ans7 = filter ans6 BY sum is null;

grunt> ans8 = foreach ans7 generate EmpId,Name;

grunt> dump ans8;

#here is the answer

(103,Akshay)

(106,Aamir)

(107,Salman)

(108,Ranbir)

(109,Katrina)

(111,Tushar)

(112,Ajay)

(113,Jubeen)