Name: Subrato Tapaswi	Class/Roll No.: D16AD/60	Grade:

Title of Experiment:
To implement the following programs using Pyspark:

1. program to find no of words starting specific letter (e.g., 'h'/'a')
2. RDBMS operations:
   - Selection
   - Projection
   - Union
   - Aggregates and grouping
   - Joins
   - Intersection

Objective of Experiment: The objective of this project is to leverage PySpark, a powerful data processing framework, to implement two distinct tasks. The first task involves developing a program to analyze text data and determine the number of words starting with specific letters. The second task focuses on performing fundamental Relational Database Management System (RDBMS) operations, including Selection, Projection, Union,Aggregates with grouping, Joins, and Intersection, using PySpark.

Outcome of Experiment: Thus, we implemented both programs using Pyspark.

Problem Statement:

a. Create a program to count words starting with specific letters (e.g., 'h' or 'a') in a given text dataset.
b. Implement key Relational Database Management System (RDBMS) operations using PySpark, including Selection, Projection, Union, Aggregates with grouping, Joins, and Intersection.

Program:

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

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

Output:

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

[(u'Subrato',1)]

Program + Output: RDD Programs

A. Selection

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

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

B. Projection

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

C. Union

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

D. Aggregate And Grouping

Sum:

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

Average:

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

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

Count

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

Max & Min Element

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

E. Join

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

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

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

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

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

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

Results and Discussions:

PySpark simplifies text analysis with word counting and mirrors RDBMS operations, including selection, projection, union, aggregates, grouping, joins, and intersection. It's a versatile tool for data manipulation, suitable for real-world applications like sentiment analysis and integration, with a focus on scalability and optimization.