**CCA175 Questions and Answers**

Question3:

Problem Scenario 57 : You have been given below code snippet. val a = sc.parallelize(1 to 9, 3) operationl  
Write a correct code snippet for operationl which will produce desired output, shown below.  
Array[(String, Seq[lnt])] = Array((even,ArrayBuffer(2, 4, G, 8)), (odd,ArrayBuffer(1, 3, 5, 7,  
9)))

Answer:

Explanation:  
Solution :  
a.groupBy(x => {if (x % 2 == 0) "even" else "odd" }).collect

Question 4:

Problem Scenario 34 : You have given a file named spark6/user.csv.  
Data is given below:  
user.csv  
id,topic,hits  
  
Rahul,scala,120 -  
  
Nikita,spark,80 -  
  
Mithun,spark,1 -  
myself,cca175,180  
Now write a Spark code in scala which will remove the header part and create RDD of values as below, for all rows. And also if id is myself" than filter out row.  
Map(id -> om, topic -> scala, hits -> 120)

Answer:

Explanation:  
Solution :  
Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.  
Step 2 : Load user.csv file from hdfs and create PairRDDs val csv = sc.textFile("spark6/user.csv")  
  
Step 3 : split and clean data -  
val headerAndRows = csv.map(line => line.split(",").map(\_.trim))  
  
Step 4 : Get header row -  
val header = headerAndRows.first  
Step 5 : Filter out header (We need to check if the first val matches the first header name) val data = headerAndRows.filter(\_(0) != header(O))  
Step 6 : Splits to map (header/value pairs)  
val maps = data.map(splits => header.zip(splits).toMap)  
step 7: Filter out the user "myself  
val result = maps.filter(map => mapf'id") != "myself")  
Step 8 : Save the output as a Text file. result.saveAsTextFile("spark6/result.txt")

Question 5:

Problem Scenario 91 : You have been given data in json format as below.  
{"first\_name":"Ankit", "last\_name":"Jain"}  
{"first\_name":"Amir", "last\_name":"Khan"}  
{"first\_name":"Rajesh", "last\_name":"Khanna"}  
{"first\_name":"Priynka", "last\_name":"Chopra"}  
{"first\_name":"Kareena", "last\_name":"Kapoor"}  
{"first\_name":"Lokesh", "last\_name":"Yadav"}  
  
Do the following activity -  
1. create employee.json tile locally.  
2. Load this tile on hdfs  
3. Register this data as a temp table in Spark using Python.  
4. Write select query and print this data.  
5. Now save back this selected data in json format.

Answer:

Explanation:  
Solution :  
Step 1 : create employee.json tile locally.  
vi employee.json (press insert) past the content.  
Step 2 : Upload this tile to hdfs, default location hadoop fs -put employee.json val employee = sqlContext.read.json("/user/cloudera/employee.json") employee.write.parquet("employee. parquet") val parq\_data = sqlContext.read.parquet("employee.parquet") parq\_data.registerTempTable("employee") val allemployee = sqlContext.sql("SELeCT' FROM employee") all\_employee.show() import org.apache.spark.sql.SaveMode prdDF.write..format("orc").saveAsTable("product ore table"}  
//Change the codec.  
sqlContext.setConf("spark.sql.parquet.compression.codec","snappy") employee.write.mode(SaveMode.Overwrite).parquet("employee.parquet")

Question 6:

Problem Scenario 2 :  
There is a parent organization called "ABC Group Inc", which has two child companies named Tech Inc and MPTech.  
Both companies employee information is given in two separate text file as below. Please do the following activity for employee details.  
  
Tech Inc.txt -  
1,Alok,Hyderabad  
2,Krish,Hongkong  
3,Jyoti,Mumbai  
4,Atul,Banglore  
5,Ishan,Gurgaon  
  
MPTech.txt -  
6,John,Newyork  
7,alp2004,California  
8,tellme,Mumbai  
9,Gagan21,Pune  
10,Mukesh,Chennai  
1. Which command will you use to check all the available command line options on HDFS and How will you get the Help for individual command.  
2. Create a new Empty Directory named Employee using Command line. And also create an empty file named in it Techinc.txt  
3. Load both companies Employee data in Employee directory (How to override existing file in HDFS).  
4. Merge both the Employees data in a Single tile called MergedEmployee.txt, merged tiles should have new line character at the end of each file content.  
5. Upload merged file on HDFS and change the file permission on HDFS merged file, so that owner and group member can read and write, other user can read the file.  
6. Write a command to export the individual file as well as entire directory from HDFS to local file System.

Answer:

Explanation:  
Solution :  
Step 1 : Check All Available command hdfs dfs  
Step 2 : Get help on Individual command hdfs dfs -help get  
Step 3 : Create a directory in HDFS using named Employee and create a Dummy file in it called e.g. Techinc.txt hdfs dfs -mkdir Employee  
Now create an emplty file in Employee directory using Hue.  
Step 4 : Create a directory on Local file System and then Create two files, with the given data in problems.  
Step 5 : Now we have an existing directory with content in it, now using HDFS command line , overrid this existing Employee directory. While copying these files from local file  
System to HDFS. cd /home/cloudera/Desktop/ hdfs dfs -put -f Employee  
Step 6 : Check All files in directory copied successfully hdfs dfs -Is Employee  
Step 7 : Now merge all the files in Employee directory, hdfs dfs -getmerge -nl Employee  
  
MergedEmployee.txt -  
Step 8 : Check the content of the file. cat MergedEmployee.txt  
Step 9 : Copy merged file in Employeed directory from local file ssytem to HDFS. hdfs dfs - put MergedEmployee.txt Employee/  
Step 10 : Check file copied or not. hdfs dfs -Is Employee  
Step 11 : Change the permission of the merged file on HDFS hdfs dfs -chmpd 664  
  
Employee/MergedEmployee.txt -  
Step 12 : Get the file from HDFS to local file system, hdfs dfs -get Employee  
  
Employee\_hdfs –

Question 7:

Problem Scenario 9 : You have been given following mysql database details as well as other info. user=retail\_dba password=cloudera database=retail\_db jdbc URL = jdbc:mysql://quickstart:3306/retail\_db  
Please accomplish following.  
1. Import departments table in a directory.  
2. Again import departments table same directory (However, directory already exist hence it should not overrride and append the results)  
3. Also make sure your results fields are terminated by '|' and lines terminated by '\n\

Answer:

Explanation:  
Solutions :  
Step 1 : Clean the hdfs file system, if they exists clean out. hadoop fs -rm -R departments hadoop fs -rm -R categories hadoop fs -rm -R products hadoop fs -rm -R orders hadoop fs -rm -R order\_items hadoop fs -rm -R customers  
Step 2 : Now import the department table as per requirement.  
sqoop import \  
-connect jdbc:mysql://quickstart:330G/retaiI\_db \  
--username=retail\_dba \  
-password=cloudera \  
-table departments \  
-target-dir=departments \  
-fields-terminated-by '|' \  
-lines-terminated-by '\n' \  
-ml  
Step 3 : Check imported data.  
hdfs dfs -Is departments  
hdfs dfs -cat departments/part-m-00000  
Step 4 : Now again import data and needs to appended.  
sqoop import \  
-connect jdbc:mysql://quickstart:3306/retail\_db \  
--username=retail\_dba \  
-password=cloudera \  
-table departments \  
-target-dir departments \  
-append \  
-tields-terminated-by '|' \  
-lines-termtnated-by '\n' \  
-ml  
Step 5 : Again Check the results  
hdfs dfs -Is departments  
hdfs dfs -cat departments/part-m-00001

Question 8:

Problem Scenario 68 : You have given a file as below.  
spark75/f ile1.txt  
File contain some text. As given Below  
spark75/file1.txt  
Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework  
The core of Apache Hadoop consists of a storage part known as Hadoop Distributed File  
System (HDFS) and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. his approach takes advantage of data locality nodes manipulating the data they have access to to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking  
For a slightly more complicated task, lets look into splitting up sentences from our documents into word bigrams. A bigram is pair of successive tokens in some sequence.  
We will look at building bigrams from the sequences of words in each sentence, and then try to find the most frequently occuring ones.  
The first problem is that values in each partition of our initial RDD describe lines from the file rather than sentences. Sentences may be split over multiple lines. The glom() RDD method is used to create a single entry for each document containing the list of all lines, we can then join the lines up, then resplit them into sentences using "." as the separator, using flatMap so that every object in our RDD is now a sentence.  
A bigram is pair of successive tokens in some sequence.

Please build bigrams from the sequences of

Answer:

Explanation:  
Solution :  
Step 1 : Create all three tiles in hdfs (We will do using Hue}. However, you can first create in local filesystem and then upload it to hdfs.  
Step 2 : The first problem is that values in each partition of our initial RDD describe lines from the file rather than sentences. Sentences may be split over multiple lines.  
The glom() RDD method is used to create a single entry for each document containing the list of all lines, we can then join the lines up, then resplit them into sentences using "." as the separator, using flatMap so that every object in our RDD is now a sentence. sentences = sc.textFile("spark75/file1.txt") \ .glom() \ map(lambda x: " ".join(x)) \ .flatMap(lambda x: x.spllt("."))  
Step 3 : Now we have isolated each sentence we can split it into a list of words and extract the word bigrams from it. Our new RDD contains tuples containing the word bigram (itself a tuple containing the first and second word) as the first value and the number 1 as the second value. bigrams = sentences.map(lambda x:x.split())  
\ .flatMap(lambda x: [((x[i],x[i+1]),1)for i in range(0,len(x)-1)])  
Step 4 : Finally we can apply the same reduceByKey and sort steps that we used in the wordcount example, to count up the bigrams and sort them in order of descending frequency. In reduceByKey the key is not an individual word but a bigram. freq\_bigrams = bigrams.reduceByKey(lambda x,y:x+y)\ map(lambda x:(x[1],x[0])) \ sortByKey(False) freq\_bigrams.take(10)

Question 9:

Problem Scenario 64 : You have been given below code snippet. val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3) val b = a.keyBy(\_.length) val c = sc.parallelize(Ust("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3) val d = c.keyBy(\_.length) operation1  
Write a correct code snippet for operational which will produce desired output, shown below.  
Array[(lnt, (Option[String], String))] = Array((6,(Some(salmon),salmon)),  
(6,(Some(salmon),rabbit}}, (6,(Some(salmon),turkey)), (6,(Some(salmon),salmon)),  
(6,(Some(salmon),rabbit)), (6,(Some(salmon),turkey)), (3,(Some(dog),dog)),  
(3,(Some(dog),cat)), (3,(Some(dog),gnu)), (3,(Some(dog),bee)), (3,(Some(rat),  
(3,(Some(rat),cat)), (3,(Some(rat),gnu)), (3,(Some(rat),bee)), (4,(None,wo!f)),  
(4,(None,bear)))

Answer:

Explanation:  
solution : b.rightOuterJqin(d).collect  
rightOuterJoin [Pair] : Performs an right outer join using two key-value RDDs. Please note that the keys must be generally comparable to make this work correctly.

Question 10:

Problem Scenario 56 : You have been given below code snippet. val a = sc.parallelize(l to 100. 3) operation1  
Write a correct code snippet for operationl which will produce desired output, shown below.  
Array [Array [I nt]] = Array(Array(1, 2, 3,4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19, 20,  
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33),  
Array(34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,  
56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66),  
Array(67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,  
89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100))

Answer:

Solution : a.glom.collect -  
glom  
Assembles an array that contains all elements of the partition and embeds it in an RDD.  
Each returned array contains the contents of one partition

Question 11:

Problem Scenario 31 : You have given following two files  
1. Content.txt: Contain a huge text file containing space separated words.  
2. Remove.txt: Ignore/filter all the words given in this file (Comma Separated).  
Write a Spark program which reads the Content.txt file and load as an RDD, remove all the words from a broadcast variables (which is loaded as an RDD of words from Remove.txt).  
And count the occurrence of the each word and save it as a text file in HDFS.  
  
Content.txt -  
  
Hello this is ABCTech.com -  
  
This is TechABY.com -  
  
Apache Spark Training -  
  
This is Spark Learning Session -  
  
Spark is faster than MapReduce -  
  
Remove.txt -  
  
Hello, is, this, the –

Answer:

Explanation:  
Solution :  
Step 1 : Create all three files in hdfs in directory called spark2 (We will do using Hue).  
However, you can first create in local filesystem and then upload it to hdfs  
Step 2 : Load the Content.txt file  
val content = sc.textFile("spark2/Content.txt") //Load the text file  
Step 3 : Load the Remove.txt file  
val remove = sc.textFile("spark2/Remove.txt") //Load the text file  
Step 4 : Create an RDD from remove, However, there is a possibility each word could have trailing spaces, remove those whitespaces as well. We have used two functions here flatMap, map and trim. val removeRDD= remove.flatMap(x=> x.splitf',") ).map(word=>word.trim)//Create an array of words  
Step 5 : Broadcast the variable, which you want to ignore  
val bRemove = sc.broadcast(removeRDD.collect().toList) // It should be array of Strings  
Step 6 : Split the content RDD, so we can have Array of String. val words = content.flatMap(line => line.split(" "))  
Step 7 : Filter the RDD, so it can have only content which are not present in "Broadcast  
Variable". val filtered = words.filter{case (word) => !bRemove.value.contains(word)}  
Step 8 : Create a PairRDD, so we can have (word,1) tuple or PairRDD. val pairRDD = filtered.map(word => (word,1))  
Step 9 : Nowdo the word count on PairRDD. val wordCount = pairRDD.reduceByKey(\_ + \_)  
Step 10 : Save the output as a Text file.  
wordCount.saveAsTextFile("spark2/result.txt")

Question 12:

Problem Scenario 3: You have been given MySQL DB with following details. user=retail\_dba password=cloudera database=retail\_db table=retail\_db.categories jdbc URL = jdbc:mysql://quickstart:3306/retail\_db  
Please accomplish following activities.  
1. Import data from categories table, where category=22 (Data should be stored in categories subset)  
2. Import data from categories table, where category>22 (Data should be stored in categories\_subset\_2)  
3. Import data from categories table, where category between 1 and 22 (Data should be stored in categories\_subset\_3)  
4. While importing catagories data change the delimiter to '|' (Data should be stored in categories\_subset\_S)  
5. Importing data from catagories table and restrict the import to category\_name,category id columns only with delimiter as '|'  
6. Add null values in the table using below SQL statement ALTER TABLE categories modify category\_department\_id int(11); INSERT INTO categories values  
(eO.NULL.'TESTING');  
7. Importing data from catagories table (In categories\_subset\_17 directory) using '|' delimiter and categoryjd between 1 and 61 and encode null values for both string and non string columns.  
8. Import entire schema retail\_db in a directory categories\_subset\_all\_tables

Answer:

Explanation:  
Solution:  
Step 1: Import Single table (Subset data} Note: Here the ' is the same you find on - key sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba - password=cloudera -table=categories ~warehouse-dir= categories\_subset --where  
\'category\_id\=22 --m 1  
Step 2 : Check the output partition  
hdfs dfs -cat categoriessubset/categories/part-m-00000  
Step 3 : Change the selection criteria (Subset data)  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba - password=cloudera -table=categories ~warehouse-dir= categories\_subset\_2 --where  
\category\_id\\>22 -m 1  
Step 4 : Check the output partition  
hdfs dfs -cat categories\_subset\_2/categories/part-m-00000  
Step 5 : Use between clause (Subset data)  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba - password=cloudera -table=categories ~warehouse-dir=categories\_subset\_3 --where  
"\category\_id\' between 1 and 22" --m 1  
Step 6 : Check the output partition  
hdfs dfs -cat categories\_subset\_3/categories/part-m-00000  
Step 7 : Changing the delimiter during import.  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail dba - password=cloudera -table=categories -warehouse-dir=:categories\_subset\_6 --where  
"/categoryjd / between 1 and 22" -fields-terminated-by='|' -m 1  
Step 8 : Check the.output partition  
hdfs dfs -cat categories\_subset\_6/categories/part-m-00000  
Step 9 : Selecting subset columns  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba - password=cloudera -table=categories --warehouse-dir=categories subset col -where  
"/category id/ between 1 and 22" -fields-terminated-by=T -columns=category name,category id --m 1  
Step 10 : Check the output partition  
hdfs dfs -cat categories\_subset\_col/categories/part-m-00000  
Step 11 : Inserting record with null values (Using mysql} ALTER TABLE categories modify category\_department\_id int(11); INSERT INTO categories values ^NULL/TESTING'); select" from categories;  
Step 12 : Encode non string null column  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail dba - password=cloudera -table=categories --warehouse-dir=categortes\_subset\_17 -where  
"\"category\_id\" between 1 and 61" -fields-terminated-by=,|' --null-string-N' -null-non- string=,N' --m 1  
  
Step 13 : View the content -  
hdfs dfs -cat categories\_subset\_17/categories/part-m-00000  
Step 14 : Import all the tables from a schema (This step will take little time) sqoop import-all-tables -connect jdbc:mysql://quickstart:3306/retail\_db -- username=retail\_dba -password=cloudera -warehouse-dir=categories\_si  
  
Step 15 : View the contents -  
hdfs dfs -Is categories\_subset\_all\_tables  
Step 16 : Cleanup or back to originals.  
delete from categories where categoryid in (59,60);  
ALTER TABLE categories modify category\_department\_id int(11) NOTNULL;  
ALTER TABLE categories modify category\_name varchar(45) NOT NULL; desc categories;

Question 13:

Problem Scenario 71 :  
Write down a Spark script using Python,  
In which it read a file "Content.txt" (On hdfs) with following content.  
After that split each row as (key, value), where key is first word in line and entire line as value.  
Filter out the empty lines.  
And save this key value in "problem86" as Sequence file(On hdfs)  
Part 2 : Save as sequence file , where key as null and entire line as value. Read back the stored sequence files.  
  
Content.txt -  
  
Hello this is ABCTECH.com -  
  
This is XYZTECH.com -  
  
Apache Spark Training -  
  
This is Spark Learning Session -  
  
Spark is faster than MapReduce –

Answer :

Explanation:  
Solution :  
Step 1 :  
# Import SparkContext and SparkConf  
from pyspark import SparkContext, SparkConf  
Step 2:  
#load data from hdfs  
contentRDD = sc.textFile(MContent.txt")  
Step 3:  
#filter out non-empty lines  
nonemptyjines = contentRDD.filter(lambda x: len(x) > 0)  
Step 4:  
#Split line based on space (Remember : It is mandatory to convert is in tuple} words = nonempty\_lines.map(lambda x: tuple(x.split('', 1))) words.saveAsSequenceFile("problem86")  
Step 5: Check contents in directory problem86 hdfs dfs -cat problem86/part\*  
Step 6 : Create key, value pair (where key is null)  
nonempty\_lines.map(lambda line: (None, Mne}).saveAsSequenceFile("problem86\_1")  
Step 7 : Reading back the sequence file data using spark. seqRDD = sc.sequenceFile("problem86\_1")  
Step 8 : Print the content to validate the same.  
for line in seqRDD.collect():  
print(line)

Question 14:

Problem Scenario 45 : You have been given 2 files , with the content as given Below  
(spark12/technology.txt)  
(spark12/salary.txt)  
(spark12/technology.txt)  
first,last,technology  
  
Amit,Jain,java -  
  
Lokesh,kumar,unix -  
  
Mithun,kale,spark -  
  
Rajni,vekat,hadoop -  
  
Rahul,Yadav,scala -  
(spark12/salary.txt)  
first,last,salary  
  
Amit,Jain,100000 -  
  
Lokesh,kumar,95000 -  
  
Mithun,kale,150000 -  
  
Rajni,vekat,154000 -  
  
Rahul,Yadav,120000 -  
Write a Spark program, which will join the data based on first and last name and save the joined results in following format, first Last.technology.salary

Answer :

Explanation:  
Solution :  
Step 1 : Create 2 files first using Hue in hdfs.  
Step 2 : Load all file as an RDD  
val technology = sc.textFile(Msparkl2/technology.txt").map(e => e.splitf',")) val salary = sc.textFile("spark12/salary.txt").map(e => e.split("."))  
Step 3 : Now create Key.value pair of data and join them.  
val joined = technology.map(e=>((e(0),e(1)),e(2))).join(salary.map(e=>((e(0),e(1)),e(2))))  
Step 4 : Save the results in a text file as below.  
joined.repartition(1).saveAsTextFile("spark12/multiColumn Joined.txt")

Question 15:

Problem Scenario 54 : You have been given below code snippet. val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle")) val b = a.map(x => (x.length, x)) operation1  
Write a correct code snippet for operationl which will produce desired output, shown below.  
Array[(lnt, String)] = Array((4,lion), (7,panther), (3,dogcat), (5,tigereagle))

Answer :

Explanation:  
Solution :  
b.foidByKey("")(\_ + J.collect  
foldByKey [Pair]  
Very similar to fold, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists of two-component tuples  
  
Listing Variants -  
def foldByKey(zeroValue: V)(func: (V, V) => V): RDD[(K, V}]  
def foldByKey(zeroValue: V, numPartitions: lnt)(func: (V, V) => V): RDD[(K, V)] def foldByKey(zeroValue: V, partitioner: Partitioner)(func: (V, V) => V): RDD[(K, V}]

Question 16:

Problem Scenario 52 : You have been given below code snippet. val b = sc.parallelize(List(1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1))  
  
Operation\_xyz -  
Write a correct code snippet for Operation\_xyz which will produce below output. scalaxollection.Map[lnt,Long] = Map(5 -> 1, 8 -> 1, 3 -> 1, 6 -> 1, 1 -> S, 2 -> 3, 4 -> 2, 7 ->  
1)

Answer :

Explanation:  
Solution :  
b.countByValue  
countByValue  
Returns a map that contains all unique values of the RDD and their respective occurrence counts. (Warning: This operation will finally aggregate the information in a single reducer.)  
  
Listing Variants -  
def countByValue(): Map[T, Long]

Question 17:

Problem Scenario 81 : You have been given MySQL DB with following details. You have been given following product.csv file product.csv productID,productCode,name,quantity,price  
1001,PEN,Pen Red,5000,1.23  
1002,PEN,Pen Blue,8000,1.25  
1003,PEN,Pen Black,2000,1.25  
1004,PEC,Pencil 2B,10000,0.48  
1005,PEC,Pencil 2H,8000,0.49  
1006,PEC,Pencil HB,0,9999.99  
Now accomplish following activities.  
1. Create a Hive ORC table using SparkSql  
2. Load this data in Hive table.  
3. Create a Hive parquet table using SparkSQL and load data in it.

Answer :

Explanation:  
Solution :  
Step 1 : Create this tile in HDFS under following directory (Without header}  
/user/cloudera/he/exam/task1/productcsv  
Step 2 : Now using Spark-shell read the file as RDD  
// load the data into a new RDD  
val products = sc.textFile("/user/cloudera/he/exam/task1/product.csv")  
// Return the first element in this RDD  
prod u cts.fi rst()  
Step 3 : Now define the schema using a case class  
case class Product(productid: Integer, code: String, name: String, quantity:lnteger, price:  
Float)  
Step 4 : create an RDD of Product objects  
val prdRDD = products.map(\_.split(",")).map(p =>  
Product(p(0).tolnt,p(1),p(2),p(3}.tolnt,p(4}.toFloat))  
prdRDD.first()  
prdRDD.count()  
Step 5 : Now create data frame val prdDF = prdRDD.toDF()  
Step 6 : Now store data in hive warehouse directory. (However, table will not be created } import org.apache.spark.sql.SaveMode prdDF.write.mode(SaveMode.Overwrite).format("orc").saveAsTable("product\_orc\_table") step 7: Now create table using data stored in warehouse directory. With the help of hive. hive show tables  
CREATE EXTERNAL TABLE products (productid int,code string,name string .quantity int, price float}  
  
STORED AS ore -  
LOCATION 7user/hive/warehouse/product\_orc\_table';  
Step 8 : Now create a parquet table  
import org.apache.spark.sql.SaveMode  
prdDF.write.mode(SaveMode.Overwrite).format("parquet").saveAsTable("product\_parquet\_ table")  
Step 9 : Now create table using this  
CREATE EXTERNAL TABLE products\_parquet (productid int,code string,name string  
.quantity int, price float}  
  
STORED AS parquet -  
LOCATION 7user/hive/warehouse/product\_parquet\_table';  
Step 10 : Check data has been loaded or not.  
Select \* from products;  
Select \* from products\_parquet;

Question 18:

Problem Scenario 19 : You have been given following mysql database details as well as other info. user=retail\_dba password=cloudera database=retail\_db jdbc URL = jdbc:mysql://quickstart:3306/retail\_db  
Now accomplish following activities.  
1. Import departments table from mysql to hdfs as textfile in departments\_text directory.  
2. Import departments table from mysql to hdfs as sequncefile in departments\_sequence directory.  
3. Import departments table from mysql to hdfs as avro file in departments avro directory.  
4. Import departments table from mysql to hdfs as parquet file in departments\_parquet directory.

Answer :

Explanation:  
Solution :  
Step 1 : Import departments table from mysql to hdfs as textfile sqoop import \  
-connect jdbc:mysql://quickstart:3306/retail\_db \  
~username=retail\_dba \  
-password=cloudera \  
-table departments \  
-as-textfile \  
-target-dir=departments\_text  
verify imported data  
hdfs dfs -cat departments\_text/part"  
Step 2 : Import departments table from mysql to hdfs as sequncetlle sqoop import \  
-connect jdbc:mysql://quickstart:330G/retaiI\_db \  
~username=retail\_dba \  
-password=cloudera \  
--table departments \  
-as-sequencetlle \  
-~target-dir=departments sequence  
verify imported data  
hdfs dfs -cat departments\_sequence/part\*  
Step 3 : Import departments table from mysql to hdfs as sequncetlle sqoop import \  
-connect jdbc:mysql://quickstart:330G/retaiI\_db \  
~username=retail\_dba \  
--password=cloudera \  
--table departments \  
--as-avrodatafile \  
--target-dir=departments\_avro  
verify imported data  
hdfs dfs -cat departments avro/part\*  
Step 4 : Import departments table from mysql to hdfs as sequncetlle sqoop import \  
-connect jdbc:mysql://quickstart:330G/retaiI\_db \  
~username=retail\_dba \  
--password=cloudera \  
-table departments \  
-as-parquetfile \  
-target-dir=departments\_parquet  
verify imported data  
hdfs dfs -cat departmentsparquet/part\*

Question 19:

Problem Scenario 58 : You have been given below code snippet.

val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2)

val b = a.keyBy(\_.length) operation1  
Write a correct code snippet for operational which will produce desired output, shown below.  
Array[(lnt, Seq[String])] = Array((4,ArrayBuffer(lion)), (6,ArrayBuffer(spider)),  
(3,ArrayBuffer(dog, cat)), (5,ArrayBuffer(tiger, eagle}}}

Answer :

Explanation:  
Solution :  
b.groupByKey.collect  
groupByKey [Pair]  
Very similar to groupBy, but instead of supplying a function, the key-component of each pair will automatically be presented to the partitioner.  
  
Listing Variants -  
def groupByKeyQ: RDD[(K, lterable[V]}]  
def groupByKey(numPartittons: Int): RDD[(K, lterable[V] )]  
def groupByKey(partitioner: Partitioner): RDD[(K, lterable[V])]

Question 20:

Problem Scenario 53 :

You have been given below code snippet. val a = sc.parallelize(1 to 10, 3) operation1 b.collect  
  
Output 1 -  
Array[lnt] = Array(2, 4, 6, 8,10)  
operation2  
  
Output 2 -  
Array[lnt] = Array(1,2, 3)  
Write a correct code snippet for operation1 and operation2 which will produce desired output, shown above.

Answer :

Explanation:  
Solution :  
valb = a.filter(\_%2==0)  
a.filter(\_ < 4).collect  
filter  
Evaluates a boolean function for each data item of the RDD and puts the items for which the function returned true into the resulting RDD.  
When you provide a filter function, it must be able to handle all data items contained in the  
RDD. Scala provides so-called partial functions to deal with mixed data types (Tip: Partial functions to deal are very useful if you have some data which may be bad and you do not want to handle but for the good data (matching data) you want to apply some Kind of map function. The following article is good. It teaches you about partial functions in a very nice way and explains why case has to be used for partial functions:article)  
Examples for mixed data without partial functions  
val b = sc.parallelize(1 to 8)  
b.filter(\_ < 4)xollect  
res15: Arrayjlnt] = Array(1, 2, 3)  
val a = sc.parallelize(List("cat'\ "horse", 4.0, 3.5, 2, "dog")) a.filter(\_<4).collect error: value < is not a member of Any

Question 21:

Problem Scenario 1:  
You have been given MySQL DB with following details.  
user=retail\_dba  
password=cloudera  
database=retail\_db  
table=retail\_db.categories  
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db  
Please accomplish following activities.  
1. Connect MySQL DB and check the content of the tables.  
2. Copy "retaildb.categories" table to hdfs, without specifying directory name.  
3. Copy "retaildb.categories" table to hdfs, in a directory name "categories\_target".  
4. Copy "retaildb.categories" table to hdfs, in a warehouse directory name  
"categories\_warehouse".

Answer :

Explanation:  
Solution :  
Step 1 : Connecting to existing MySQL Database mysql --user=retail\_dba -- password=cloudera retail\_db  
Step 2 : Show all the available tables show tables;  
Step 3 : View/Count data from a table in MySQL select count(1} from categories;  
Step 4 : Check the currently available data in HDFS directory hdfs dfs -Is  
Step 5 : Import Single table (Without specifying directory).  
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db -username=retail\_dba - password=cloudera -table=categories  
Note : Please check you dont have space between before or after '=' sign. Sqoop uses the  
MapReduce framework to copy data from RDBMS to hdfs  
Step 6 : Read the data from one of the partition, created using above command, hdfs dfs - catxategories/part-m-00000  
Step 7 : Specifying target directory in import command (We are using number of mappers  
=1, you can change accordingly) sqoop import -connect  
jdbc:mysql://quickstart:3306/retail\_db -username=retail\_dba -password=cloudera  
~table=categories -target-dir=categortes\_target --m 1  
Step 8 : Check the content in one of the partition file.  
hdfs dfs -cat categories\_target/part-m-00000  
Step 9 : Specifying parent directory so that you can copy more than one table in a specified target directory. Command to specify warehouse directory. sqoop import -.-connect jdbc:mysql://quickstart:3306/retail\_db --username=retail dba - password=cloudera -table=categories -warehouse-dir=categories\_warehouse --m 1

Question 22:

Problem Scenario 63 : You have been given below code snippet. val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2) val b = a.map(x => (x.length, x)) operation1  
Write a correct code snippet for operationl which will produce desired output, shown below.  
Array[(lnt, String}] = Array((4,lion), (3,dogcat), (7,panther), (5,tigereagle))

Answer :

Explanation:  
Solution :  
b.reduceByKey(\_ + \_).collect  
reduceByKey JPair] : This function provides the well-known reduce functionality in Spark.  
Please note that any function f you provide, should be commutative in order to generate reproducible results.

Question 23:

Problem Scenario 89 : You have been given below patient data in csv format, patientID,name,dateOfBirth,lastVisitDate  
1001,Ah Teck,1991-12-31,2012-01-20  
1002,Kumar,2011-10-29,2012-09-20  
1003,Ali,2011-01-30,2012-10-21  
Accomplish following activities.  
1. Find all the patients whose lastVisitDate between current time and '2012-09-15'  
2. Find all the patients who born in 2011  
3. Find all the patients age  
4. List patients whose last visited more than 60 days ago  
5. Select patients 18 years old or younger

Answer :

Explanation:  
Solution :  
Step 1:  
hdfs dfs -mkdir sparksql3  
hdfs dfs -put patients.csv sparksql3/  
  
Step 2 : Now in spark shell -  
// SQLContext entry point for working with structured data  
val sqlContext = neworg.apache.spark.sql.SQLContext(sc)  
// this is used to implicitly convert an RDD to a DataFrame.  
import sqlContext.impIicits.\_  
// Import Spark SQL data types and Row.  
import org.apache.spark.sql.\_  
// load the data into a new RDD  
val patients = sc.textFilef'sparksqIS/patients.csv")  
// Return the first element in this RDD  
patients.first()  
//define the schema using a case class  
case class Patient(patientid: Integer, name: String, dateOfBirth:String , lastVisitDate:  
String)  
// create an RDD of Product objects  
val patRDD = patients.map(\_.split(M,M)).map(p => Patient(p(0).tolnt,p(1),p(2),p(3))) patRDD.first() patRDD.count(}  
// change RDD of Product objects to a DataFrame val patDF = patRDD.toDF()  
// register the DataFrame as a temp table patDF.registerTempTable("patients"}  
// Select data from table  
val results = sqlContext.sql(......SELECT\* FROM patients '.....)  
// display dataframe in a tabular format  
results.show()  
//Find all the patients whose lastVisitDate between current time and '2012-09-15' val results = sqlContext.sql(......SELECT \* FROM patients WHERE  
TO\_DATE(CAST(UNIX\_TIMESTAMP(lastVisitDate, 'yyyy-MM-dd') AS TIMESTAMP))  
BETWEEN '2012-09-15' AND current\_timestamp() ORDER BY lastVisitDate......) results.showQ  
/.Find all the patients who born in 2011  
val results = sqlContext.sql(......SELECT \* FROM patients WHERE  
YEAR(TO\_DATE(CAST(UNIXJTlMESTAMP(dateOfBirth, 'yyyy-MM-dd') AS  
TIMESTAMP))) = 2011 ......)  
results. show()  
//Find all the patients age  
val results = sqlContext.sql(......SELECT name, dateOfBirth, datediff(current\_date(),  
TO\_DATE(CAST(UNIX\_TIMESTAMP(dateOfBirth, 'yyyy-MM-dd') AS TlMESTAMP}}}/365  
  
AS age -  
  
FROM patients -  
Mini >  
results.show()  
//List patients whose last visited more than 60 days ago  
-- List patients whose last visited more than 60 days ago  
val results = sqlContext.sql(......SELECT name, lastVisitDate FROM patients WHERE datediff(current\_date(), TO\_DATE(CAST(UNIX\_TIMESTAMP[lastVisitDate, 'yyyy-MM-dd')  
AS T1MESTAMP))) > 60......);  
results. showQ;  
-- Select patients 18 years old or younger  
SELECT' FROM patients WHERE TO\_DATE(CAST(UNIXJTlMESTAMP(dateOfBirth,  
'yyyy-MM-dd') AS TIMESTAMP}) > DATE\_SUB(current\_date(),INTERVAL 18 YEAR); val results = sqlContext.sql(......SELECT' FROM patients WHERE  
TO\_DATE(CAST(UNIX\_TIMESTAMP(dateOfBirth, 'yyyy-MM--dd') AS TIMESTAMP)) >  
DATE\_SUB(current\_date(), T8\*365)......);  
results. showQ;  
val results = sqlContext.sql(......SELECT DATE\_SUB(current\_date(), 18\*365) FROM patients......); results.show();

Question 24:

Problem Scenario 48 : You have been given below Python code snippet, with intermediate output.  
We want to take a list of records about people and then we want to sum up their ages and count them.  
So for this example the type in the RDD will be a Dictionary in the format of {name: NAME, age:AGE, gender:GENDER}.  
The result type will be a tuple that looks like so (Sum of Ages, Count) people = [] people.append({'name':'Amit', 'age':45,'gender':'M'}) people.append({'name':'Ganga', 'age':43,'gender':'F'}) people.append({'name':'John', 'age':28,'gender':'M'}) people.append({'name':'Lolita', 'age':33,'gender':'F'}) people.append({'name':'Dont Know', 'age':18,'gender':'T'}) peopleRdd=sc.parallelize(people) //Create an RDD peopleRdd.aggregate((0,0), seqOp, combOp) //Output of above line : 167, 5)  
Now define two operation seqOp and combOp , such that  
seqOp : Sum the age of all people as well count them, in each partition. combOp :  
Combine results from all partitions.

Answer :

Explanation:  
Solution :  
seqOp = (lambda x,y: (x[0] + y['age'],x[1] + 1))  
combOp = (lambda x,y: (x[0] + y[0], x[1] + y[1]))