DUMPSARENA

# CCA Spark and Hadoop Developer Exam

## Cloudera CCA175

**Version Demo**

**Total Demo Questions: 10**

**Total Premium Questions: 96**

**Buy Premium PDF**

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

## QUESTION NO: 1 - (SIMULATION)

SIMULATION9

SIMULATION

Problem Scenario 79 : You have been given MySQL DB with following details.

user=retail_dba

password=cloudera

database=retail_db

table=retail_db.orders

table=retail_db.order_items

jdbc URL = jdbc:mysql://quickstart:3306/retail_db

Columns of products table : (product_id | product categoryid | product_name | product_description | product_prtce | product_image )

Please accomplish following activities.

1. Copy "retaildb.products" table to hdfs in a directory p93_products

2. Filter out all the empty prices

3. Sort all the products based on price in both ascending as well as descending order.

4. Sort all the products based on price as well as product_id in descending order.

5. Use the below functions to do data ordering or ranking and fetch top 10 elements top()

takeOrdered() sortByKey()

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution :

Step 1 : Import Single table .

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -table=products -target-dir=p93_products -m 1

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 2 : Step 2 : Read the data from one of the partition, created using above command, hadoop fs -cat p93_products/part-m-00000

Step 3 : Load this directory as RDD using Spark and Python (Open pyspark terminal and do following). productsRDD = sc.textFile("p93_products")

Step 4 : Filter empty prices, if exists

#filter out empty prices lines

nonemptyjines = productsRDD.filter(lambda x: len(x.split(",")[4]) > 0)

Step 5 : Now sort data based on product_price in order. sortedPriceProducts=nonempty_lines.map(lambdaline:(float(line.split(",")[4]),line.split(",")[2])).sortByKey()

for line in sortedPriceProducts.collect(): print(line)

Step 6 : Now sort data based on product_price in descending order. sortedPriceProducts=nonempty_lines.map(lambda line: (float(line.split(",")[4]),line.split(",")[2])).sortByKey(False)

for line in sortedPriceProducts.collect(): print(line)

Step 7 : Get highest price products name. sortedPriceProducts=nonemptyJines.map(lambda line : (float(line.split(",")[4]),line-split(,,,,,)[2]))-sortByKey(False).take(1)

print(sortedPriceProducts)

Step 8 : Now sort data based on product_price as well as product_id in descending order.

#Dont forget to cast string #Tuple as key ((price,id),name)

sortedPriceProducts=nonemptyJines.map(lambda line : ((float(line print(sortedPriceProducts)

Step 9 : Now sort data based on product_price as well as product_id in descending order, using top() function.

#Dont forget to cast string

#Tuple as key ((price,id),name)

sortedPriceProducts=nonemptyJines.map(lambda line: ((float(line.s^^

print(sortedPriceProducts)

Step 10 : Now sort data based on product_price as ascending and product_id in ascending order, using takeOrdered{) function.

#Dont forget to cast string

#Tuple as key ((price,id),name) sortedPriceProducts=nonemptyJines.map(lambda line: ((float(line.split(","}[4]},int(line.split(","}[0]}},line.split(","}[2]}}.takeOrdered(10, lambda tuple : (tuple[0][0],tuple[0][1]))

Step 11 : Now sort data based on product_price as descending and product_id in ascending order, using takeOrdered() function.

#Dont forget to cast string

#Tuple as key ((price,id},name)

#Using minus(-) parameter can help you to make descending ordering , only for numeric value.

sortedPrIceProducts=nonemptylines.map(lambda line: ((float(line.split(","}[4]},int(line.split(","}[0]}},line.split(","}[2]}}.takeOrdered(10, lambda tuple : (-tuple[0][0],tuple[0][1]}}

## QUESTION NO: 2 - (SIMULATION)

Problem Scenario 92 : You have been given a spark scala application, which is bundled in jar named hadoopexam.jar.

Your application class name is com.hadoopexam.MyTask

You want that while submitting your application should launch a driver on one of the cluster node.

Please complete the following command to submit the application.

spark-submit XXX -master yarn \

YYY SSPARK HOME/lib/hadoopexam.jar 10

**ANSWER: See the explanation for Step by Step Solution and configuration**

**Explanation:**

Solution

XXX: -class com.hadoopexam.MyTask

YYY : --deploy-mode cluster

## QUESTION NO: 3 - (SIMULATION)

SIMULATION

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: See the explanation for Step by Step Solution and configuration.**

**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 -ls 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 -ls 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 NO: 4 - (SIMULATION)

SIMULATION5

SIMULATION

Problem Scenario 55 : You have been given below code snippet.

val pairRDDI = sc.parallelize(List( ("cat",2), ("cat", 5), ("book", 4),("cat", 12))) val pairRDD2 = sc.parallelize(List( ("cat",2), ("cup", 5), ("mouse", 4),("cat", 12)))

operation1

Write a correct code snippet for operationl which will produce desired output, shown below.

Array[(String, (Option[lnt], Option[lnt]))] = Array((book,(Some(4},None)), (mouse,(None,Some(4))), (cup,(None,Some(5))), (cat,(Some(2),Some(2)), (cat,(Some(2),Some(12))), (cat,(Some(5),Some(2))), (cat,(Some(5),Some(12))), (cat,(Some(12),Some(2))), (cat,(Some(12),Some(12)))J

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution : pairRDD1.fullOuterJoin(pairRDD2).collect

fullOuterJoin [Pair]

Performs the full outer join between two paired RDDs.

Listing Variants

def fullOuterJoin[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Option[V], OptionfW]))]

def fullOuterJoin[W](other: RDD[(K, W}]}: RDD[(K, (Option[V], OptionfW]))]

def fullOuterJoin[W](other: RDD[(K, W)], partitioner: Partitioner): RDD[(K, (Option[V], Option[W]))]

## QUESTION NO: 5 - (SIMULATION)

SIMULATION7

SIMULATION

Problem Scenario 37 : ABCTECH.com has done survey on their roducts feedback using a web based form. With the following free text field as input in web ui.

Name: String

Subscription Date: String

Rating : String

And servey data has been saved in a file called spark9/feedback.txt

Christopher|Jan 11, 2015|5

Kapil|11 Jan, 2015|5

Thomas|6/17/2014|5

John|22-08-2013|5

Mithun|2013|5

Jitendra||5

Write a spark program using regular expression which will filter all the valid dates and save in two separate file (good record and bad record)

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution :

Step 1 : Create a file first using Hue in hdfs.

Step 2 : Write all valid regular expressions sysntex for checking whether records are having valid dates or not.

val regl =.....(\d+)\s(\w{3})(,)\s(\d{4}).....r//11 Jan, 2015

val reg2 =.....(\d+)(U)(\d+)(U)(\d{4}).....s II 6/17/2014

val reg3 =.....(\d+)(-)(\d+)(-)(\d{4})""".r//22-08-2013

val reg4 =.....(\w{3})\s(\d+)(,)\s(\d{4}).....s II Jan 11, 2015

Step 3 : Load the file as an RDD.

val feedbackRDD = sc.textFile("spark9/feedback.txt"}

Step 4 : As data are pipe separated , hence split the same. val feedbackSplit = feedbackRDD.map(line => line.split('|'))

Step 5 : Now get the valid records as well as , bad records.

val validRecords = feedbackSplit.filter(x =>
(reg1.pattern.matcher(x(1).trim).matches|reg2.pattern.matcher(x(1).trim).matches|reg3.pattern.matcher(x(1).trim).matches |
reg4.pattern.matcher(x(1).trim).matches))

val badRecords = feedbackSplit.filter(x =>
!(reg1.pattern.matcher(x(1).trim).matches|reg2.pattern.matcher(x(1).trim).matches|reg3.pattern.matcher(x(1).trim).matches |
reg4.pattern.matcher(x(1).trim).matches))

Step 6 : Now convert each Array to Strings

val valid =vatidRecords.map(e => (e(0),e(1),e(2)))

val bad =badRecords.map(e => (e(0),e(1),e(2)))

Step 7 : Save the output as a Text file and output must be written in a single tile,

valid.repartition(1).saveAsTextFile("spark9/good.txt") bad.repartition(1).saveAsTextFile("sparkS7bad.txt")

**QUESTION NO: 6 - (SIMULATION)**

SIMULATION3

SIMULATION

Problem Scenario 83 : In Continuation of previous question, please accomplish following activities.

1. Select all the records with quantity >= 5000 and name starts with 'Pen'

2. Select all the records with quantity >= 5000, price is less than 1.24 and name starts with 'Pen'

3. Select all the records witch does not have quantity >= 5000 and name does not starts with 'Pen'

4. Select all the products which name is 'Pen Red', 'Pen Black'

5. Select all the products which has price BETWEEN 1.0 AND 2.0 AND quantity BETWEEN 1000 AND 2000.

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution :

Step 1 : Select all the records with quantity >= 5000 and name starts with 'Pen'

val results = sqlContext.sql(.....SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %.....)

results.show()

Step 2 : Select all the records with quantity >= 5000 , price is less than 1.24 and name starts with 'Pen'

val results = sqlContext.sql(.....SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %.....)

results. showQ

Step 3 : Select all the records witch does not have quantity >= 5000 and name does not starts with 'Pen'

val results = sqlContext.sql('.....SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %').....)

results. showQ

Step 4 : Select all the products wchich name is 'Pen Red', 'Pen Black'

val results = sqlContext.sql('.....SELECT' FROM products WHERE name IN ('Pen Red', 'Pen Black').....)

results. showQ

Step 5 : Select all the products which has price BETWEEN 1.0 AND 2.0 AND quantity BETWEEN 1000 AND 2000.

val results = sqlContext.sql(.....SELECT * FROM products WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000).....)

results. show()

**QUESTION NO: 7 - (SIMULATION)**

SIMULATION8

SIMULATION

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 operationl 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: See the explanation for Step by Step Solution and configuration.**

**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 NO: 8 - (SIMULATION)

SIMULATION2

SIMULATION

Problem Scenario 62 : You have been given below code snippet.

val a = sc.parallelize(List("dogM, "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((3,xdogx), (5,xtigerx), (4,xlionx), (3,xcatx), (7,xpantherx), (5,xeaglex))

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution :

b.mapValuesf'x" + _ + "x").collect

mapValues [Pair] : Takes the values of a RDD that consists of two-component tuples, and applies the provided function to transform each value. Tlien,.it.forms newtwo-componend tuples using the key and the transformed value and stores them in a new RDD.

## QUESTION NO: 9 - (SIMULATION)

SIMULATION7

SIMULATION

Problem Scenario 87 : You have been given below three files

product.csv (Create this file in hdfs)

productID,productCode,name,quantity,price,supplierid

1001,PEN,Pen Red,5000,1.23,501

1002,PEN,Pen Blue,8000,1.25,501

1003,PEN,Pen Black,2000,1.25,501

1004,PEC,Pencil 2B,10000,0.48,502

1005,PEC,Pencil 2H,8000,0.49,502

1006,PEC,Pencil HB,0,9999.99,502

2001,PEC,Pencil 3B,500,0.52,501

2002,PEC,Pencil 4B,200,0.62,501

2003,PEC,Pencil 5B,100,0.73,501

2004,PEC,Pencil 6B,500,0.47,502

supplier.csv

supplierid,name,phone

501,ABC Traders,88881111

502,XYZ Company,88882222

503,QQ Corp,88883333

products_suppliers.csv

productID,supplierID

2001,501

2002,501

2003,501

2004,502

2001,503

Now accomplish all the queries given in solution.

Select product, its price , its supplier name where product price is less than 0.6 using SparkSQL

**ANSWER: See the explanation for Step by Step Solution and configuration.**

**Explanation:**

Solution :

Step 1:

hdfs dfs -mkdir sparksql2

hdfs dfs -put product.csv sparksq!2/

hdfs dfs -put supplier.csv sparksql2/

hdfs dfs -put products_suppliers.csv sparksql2/

Step 2 : Now in spark shell

// this Is used to Implicitly convert an RDD to a DataFrame.

import sqlContext.implicits._

// Import Spark SQL data types and Row.

import org.apache.spark.sql._

// load the data into a new RDD

val products = sc.textFile("sparksql2/product.csv")

val supplier = sc.textFileC'sparksq^supplier.csv")

val prdsup = sc.textFile("sparksql2/products_suppliers.csv"}

// Return the first element in this RDD

products.fi rst()

supplier.first{).

prdsup.first()

//define the schema using a case class

case class Product(productid: Integer, code: String, name: String, quantity:Integer, price: Float, supplierid:Integer)

case class Suplier(supplierid: Integer, name: String, phone: String)

case class PRDSUP(productid: Integer.supplierid: Integer)

// create an RDD of Product objects

val prdRDD = products.map(_.split("\")).map(p => Product(p(0).toInt,p(1),p(2),p(3).toInt,p(4).toFloat,p(5).toint))

val supRDD = supplier.map(_.split(",")).map(p => Suplier(p(0).toInt,p(1),p(2)))

val prdsupRDD = prdsup.map(_.split(",")).map(p => PRDSUP(p(0).toInt,p(1}.toInt}}

prdRDD.first()

prdRDD.count()

supRDD.first() supRDD.count()

prdsupRDD.first() prdsupRDD.count{}

// change RDD of Product objects to a DataFrame

val prdDF = prdRDD.toDF()

val supDF = supRDD.toDF()

val prdsupDF = prdsupRDD.toDF()

// register the DataFrame as a temp table prdDF.registerTempTablef'products")

supDF.registerTempTablef'suppliers")

prdsupDF.registerTempTablef'productssuppliers"}

//Select product, its price , its supplier name where product price is less than 0.6

val results = sqlContext.sql(.....SELECT products.name, price, suppliers.name as sup_name FROM products JOIN suppliers ON products.supplierlD= suppliers.supplierlD WHERE price < 0.6.....]

results. show()

## QUESTION NO: 10 - (SIMULATION)

SIMULATION2

SIMULATION

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: See the explanation for Step by Step Solution and configuration.**

**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]