CCA175 Q&As

CCA Spark and Hadoop Developer Exam

Pass Cloudera CCA175 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.leads4pass.com/cca175.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Cloudera Official Exam Center

QUESTION 1

Problem Scenario 93 : You have to run your Spark application with locally 8 thread or locally on 8 cores. Replace XXX with correct values. spark-submit --class com.hadoopexam.MyTask XXX \ -deploy-mode cluster SSPARK_HOME/lib/hadoopexam.jar 10

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution

XXX: -master local[8]

Notes : The master URL passed to Spark can be in one of the following formats:

Master URL Meaning

local Run Spark locally with one worker thread (i.e. no parallelism at all}.

local[K] Run Spark locally with K worker threads (ideally, set this to the number of cores on

your machine).

local[*] Run Spark locally with as many worker threads as logical cores on your machine.

spark://HOST:PORT Connect to the given Spark standalone cluster master. The port must

be whichever one your master is configured to use, which is 7077 by default.

mesos://HOST:PORT Connect to the given Mesos cluster. The port must be whichever one

your is configured to use, which is 5050 by default. Or, for a Mesos cluster using

ZooKeeper, use mesos://zk://.... To submit with --deploy-mode cluster, the HOST:PORT

should be configured to connect to the MesosClusterDispatcher.

yarn Connect to a YARN cluster in client or cluster mode depending on the value of deploy-mode. The cluster location will be found based on the HADOOP CONF DIR or

YARN CONF DIR variable.

QUESTION 2

Problem Scenario 39 : You have been given two files spark16/file1.txt 1,9,5 2,7,4 3,8,3 spark16/file2.txt 1,g,h 2,i,j 3,k,l Load these two tiles as Spark RDD and join them to produce the below results (l,((9,5),(g,h))) (2, ((7,4), (i,j))) (3, ((8,3), (k,l))) And write code snippet which will sum the second columns of above joined results (5+4+3).

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

Step 1 : Create tiles in hdfs using Hue.

Step 2 : Create pairRDD for both the files.

val one = sc.textFile("spark16/file1.txt").map{

_.split(",",-1) match {

case Array(a, b, c) => (a, (b, c))

} }

val two = sc.textFHe(Mspark16/file2.txt").map{

_.split(\\'7\-1) match {

case Array(a, b, c) => (a, (b, c))

} }

Step 3 : Join both the RDD. val joined = one.join(two)

Step 4 : Sum second column values.

val sum = joined.map {

case (_, ((_, num2), (_, _))) => num2.tolnt

}.reduce(_ + _)

QUESTION 3

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.

Correct Answer: See the explanation for Step by Step Solution and configuration.

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 4

Problem Scenario 86 : In Continuation of previous question, please accomplish following activities.

1.

Select Maximum, minimum, average , Standard Deviation, and total quantity.

2.

Select minimum and maximum price for each product code.

3.

Select Maximum, minimum, average , Standard Deviation, and total quantity for each product code, hwoever make sure Average and Standard deviation will have maximum two decimal values.

4.

Select all the product code and average price only where product count is more than or equal to 3.

5.

Select maximum, minimum , average and total of all the products for each code. Also produce the same across all the products.

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

Step 1 : Select Maximum, minimum, average , Standard Deviation, and total quantity.

val results = sqlContext.sql(\\'.....SELECT MAX(price) AS MAX , MIN(price) AS MIN ,

AVG(price) AS Average, STD(price) AS STD, SUM(quantity) AS total_products FROM

products.....)

results. showQ

Step 2 : Select minimum and maximum price for each product code.

val results = sqlContext.sql(.....SELECT code, MAX(price) AS Highest Price\\', MIN(price)

AS Lowest Price\\'

FROM products GROUP BY code.....)

results. showQ

Step 3 : Select Maximum, minimum, average , Standard Deviation, and total quantity for

each product code, hwoever make sure Average and Standard deviation will have

maximum two decimal values.

val results = sqlContext.sql(.....SELECT code, MAX(price), MIN(price),

CAST(AVG(price} AS DECIMAL(7,2)) AS Average\\', CAST(STD(price) AS DECIMAL(7,2))

AS \\'Std Dev\ SUM(quantity) FROM products

GROUP BY code.....)

results. showQ

Step 4 : Select all the product code and average price only where product count is more

than or equal to 3.

val results = sqlContext.sql(.....SELECT code AS Product Code\\',

COUNTf) AS Count\\',

CAST(AVG(price) AS DECIMAL(7,2)) AS Average\\' FROM products GROUP BY code

HAVING Count >=3"M") results. showQ

Step 5 : Select maximum, minimum , average and total of all the products for each code.

Also produce the same across all the products.

val results = sqlContext.sql("""SELECT

code,

MAX(price),

MIN(pnce),

CAST(AVG(price) AS DECIMAL(7,2)) AS Average\\',

SUM(quantity)FROM products

GROUP BY code

WITH ROLLUP""")

results. show()

QUESTION 5

Problem Scenario 96 : Your spark application required extra Java options as below. XX:+PrintGCDetails-XX:+PrintGCTimeStamps Please replace the XXX values correctly ./bin/spark-submit --name "My app" --master local[4] --conf spark.eventLog.enabled=talse -conf XXX hadoopexam.jar

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution

XXX: Mspark.executoi\extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps"

Notes: ./bin/spark-submit \

--class

--master \

--deploy-mode \

-conf = \

other options

\

[application-arguments]

Here, conf is used to pass the Spark related contigs which are required for the application

to run like any specific property(executor memory) or if you want to override the default

property which is set in Spark-default.conf.