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.pass2lead.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 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()

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

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]),linesplit(,,,,,)[

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.

sortedPrlceProducts=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 2

Problem Scenario 65 : You have been given below code snippet.

val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2)

val b = sc.parallelize(1 to a.count.tolnt, 2)

val c = a.zip(b)

operation1

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

Array[(String, Int)] = Array((owl,3), (gnu,4), (dog,1), (cat,2>, (ant,5))

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

Solution : c.sortByKey(false).collect

sortByKey [Ordered] : This function sorts the input RDD\\'s data and stores it in a new RDD.

"The output RDD is a shuffled RDD because it stores data that is output by a reducer

which has been shuffled. The implementation of this function is actually very clever.

First, it uses a range partitioner to partition the data in ranges within the shuffled RDD.

Then it sorts these ranges individually with mapPartitions using standard sort mechanisms.

QUESTION 3

Problem Scenario 5 : You have been given following mysql database details.

user=retail_dba password=cloudera database=retail_db jdbc URL = jdbc:mysql://quickstart:3306/retail_db Please accomplish following activities.

1.

List all the tables using sqoop command from retail_db

2.

Write simple sqoop eval command to check whether you have permission to read database tables or not.

3.

Import all the tables as avro files in /user/hive/warehouse/retail cca174.db

4.

Import departments table as a text file in /user/cloudera/departments.

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

Solution:

Step 1 : List tables using sqoop

sqoop list-tables --connect jdbc:mysql://quickstart:330G/retail_db --username retail dba password cloudera

Step 2 : Eval command, just run a count query on one of the table.

sqoop eval \

--connect jdbc:mysql://quickstart:3306/retail_db \

-username retail_dba \

-password cloudera \

--query "select count(1) from ordeMtems"

Step 3 : Import all the tables as avro file.

sqoop import-all-tables \

-connect jdbc:mysql://quickstart:3306/retail_db \

-username=retail_dba \

-password=cloudera \

-as-avrodatafile \

-warehouse-dir=/user/hive/warehouse/retail stage.db \

-ml

Step 4 : Import departments table as a text file in /user/cloudera/departments

sqoop import \

-connect jdbc:mysql://quickstart:3306/retail_db \

-username=retail_dba \

-password=cloudera \

-table departments \

-as-textfile \

-target-dir=/user/cloudera/departments

Step 5 : Verify the imported data.

hdfs dfs -ls /user/cloudera/departments

hdfs dfs -ls /user/hive/warehouse/retailstage.db

hdfs dfs -ls /user/hive/warehouse/retail_stage.db/products

QUESTION 4

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

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

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.implicits._

// 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(UNIXJTIMESTAMP(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 TIMESTAMP}}}/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(UNIXJTIMESTAMP(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 5

Problem Scenario 61 : 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(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3)

val d = c.keyBy(_.length) operationl

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

Array[(lnt, (String, Option[String]}}] = Array((6,(salmon,Some(salmon))),

(6,(salmon,Some(rabbit))),

(6,(salmon,Some(turkey))), (6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))),

(6,(salmon,Some(turkey))), (3,(dog,Some(dog))), (3,(dog,Some(cat))),

(3,(dog,Some(dog))), (3,(dog,Some(bee))), (3,(rat,Some(dogg)), (3,(rat,Some(cat)j),

(3,(rat.Some(gnu))). (3,(rat,Some(bee))), (8,(elephant,None)))

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

Solution : b.leftOuterJoin(d}.collect leftOuterJoin [Pair]: Performs an left outer join using two key-value RDDs. Please note that the keys must be generally comparable to make this work keyBy : Constructs twocomponent tuples (key-value pairs) by applying a function on each data item. Trie result of the function becomes the key and the original data item becomes the value of the newly created tuples.