

CCA Spark and Hadoop Developer Exam

Cloudera CCA175 Dumps Available Here at:

<https://www.certification-questions.com/cloudera-exam/cca175-dumps.html>

Enrolling now you will get access to 93 questions in a unique set of CCA175 dumps

Question 1

CORRECT TEXT

Problem Scenario 49 : You have been given below code snippet (do a sum of values by key}, with intermediate output.

```
val keysWithValuesList = Array("foo=A", "foo=A", "foo=A", "foo=A", "foo=B", "bar=C", "bar=D", "bar=D")
```

```
val data = sc.parallelize(keysWithValuesList)
```

```
//Create key value pairs
```

```
val kv = data.map(_.split("=")).map(v => (v(0), v(1))).cache()
```

```
val initialCount = 0;
```

```
val countByKey = kv.aggregateByKey(initialCount)(addToCounts, sumPartitionCounts)
```

Now define two functions (addToCounts, sumPartitionCounts) such, which will produce following results.

Output 1

```
countByKey.collect
```

```
res3: Array[(String, Int)] = Array((foo,5), (bar,3))
```

```
import scala.collection._
```

```
val initialSet = scala.collection.mutable.HashSet.empty[String]
```

```
val uniqueByKey = kv.aggregateByKey(initialSet)(addToSet, mergePartitionSets)
```

Now define two functions (addToSet, mergePartitionSets) such, which will produce following results.

Output 2:

```
uniqueByKey.collect
```

```
res4: Array[(String, scala.collection.mutable.HashSet[String])] = Array((foo,Set(B, A)), (bar,Set(C, D)))
```

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

```
val addToCounts = (n: Int, v: String) => n + 1
val sumPartitionCounts = (p1: Int, p2: Int) => p1 + p2
val addToSet = (s: mutable.HashSet[String], v: String) => s += v
val mergePartitionSets = (p1: mutable.HashSet[String], p2: mutable.HashSet[String]) => p1
+ += p2
```

Answer: A

Explanation:

rrrect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

```
val addToCounts = (n: Int, v: String) => n + 1
val sumPartitionCounts = (p1: Int, p2: Int) => p1 + p2
val addToSet = (s: mutable.HashSet[String], v: String) => s += v
val mergePartitionSets = (p1: mutable.HashSet[String], p2: mutable.HashSet[String]) => p1
+ += p2
```

Question 2

CORRECT TEXT

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.

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Create this file in HDFS under following directory (Without header)

/user/cloudera/he/exam/task1/product.csv

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

```
products.first()
```

Step 3 : Now define the schema using a case class

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

Step 4 : 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))
```

```
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 orc
```

```
LOCATION /user/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 /user/hive/warehouse/product_parquet_table';
```

Step 10 : Check data has been loaded or not.

```
Select * from products;
```

```
Select * from products_parquet;
```

Answer: A**Explanation:**

rect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Create this file 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

```
products.first()
```

Step 3 : Now define the schema using a case class

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

Step 4 : 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))
```

```
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 orc
```

```
LOCATION 'user/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 'user/hive/warehouse/product_parquet_table';
```

Step 10 : Check data has been loaded or not.

```
Select * from products;
```

Select * from products_parquet;

Question 3

CORRECT TEXT

Problem Scenario 84 : In Continuation of previous question, please accomplish following activities.

1. Select all the products which has product code as null
2. Select all the products, whose name starts with Pen and results should be order by Price descending order.

3. Select all the products, whose name starts with Pen and results should be order by Price descending order and quantity ascending order.
4. Select top 2 products by price

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Select all the products which has product code as null

```
val results = sqlContext.sql(.....SELECT' FROM products WHERE code IS NULL.....) results.  
showQ val
```

```
results = sqlContext.sql(.....SELECT * FROM products WHERE code = NULL ",,M )  
results.showQ
```

Step 2 : Select all the products , whose name starts with Pen and results should be order by Price descending order. val results = sqlContext.sql(.....SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC.....) results. showQ

Step 3 : Select all the products , whose name starts with Pen and results should be order by Price descending order and quantity ascending order. val results = sqlContext.sql('.....SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity.....) results. showQ

Step 4 : Select top 2 products by price

```
val results = sqlContext.sql(.....SELECT' FROM products ORDER BY price desc  
LIMIT2.....}  
results. show()
```

Answer: A

Explanation:

rect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Select all the products which has product code as null

val results = sqlContext.sql(.....SELECT' FROM products WHERE code IS NULL.....) results. showQ val

results = sqlContext.sql(.....SELECT * FROM products WHERE code = NULL ",,M) results.showQ

Step 2 : Select all the products , whose name starts with Pen and results should be order by Price

descending order. val results = sqlContext.sql(.....SELECT * FROM products

WHERE name LIKE 'Pen %' ORDER BY price DESC.....)

results. showQ

Step 3 : Select all the products , whose name starts with Pen and results should be order by Price

descending order and quantity ascending order. val results = sqlContext.sql('.....SELECT * FROM

products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity.....) results. showQ

Step 4 : Select top 2 products by price

val results = sqlContext.sql(.....SELECT' FROM products ORDER BY price desc

LIMIT2.....}

results. show()

Question 4

CORRECT TEXT

Problem Scenario 4: 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.

Import Single table categories (Subset data) to hive managed table , where category_id between 1 and 22

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Import Single table (Subset data)

```
sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -  
password=cloudera -table=categories -where "'category_id\' between 1 and 22" --hive- import --m  
1
```

Note: Here the ' is the same you find on ~ key

This command will create a managed table and content will be created in the following directory.

/user/hive/warehouse/categories

Step 2 : Check whether table is created or not (In Hive)

show tables;

select * from categories;

Answer: A

Explanation:

rrrect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Import Single table (Subset data)

```
sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -  
password=cloudera -table=categories -where "'category_id\' between 1 and 22" --hive- import --m 1
```

Note: Here the ' is the same you find on ~ key

This command will create a managed table and content will be created in the following directory.

/user/hive/warehouse/categories

Step 2 : Check whether table is created or not (In Hive)

show tables;

select * from categories;

Question 5

CORRECT TEXT

Problem Scenario 13 : 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. Create a table in retaildb with following definition.

```
CREATE table departments_export (department_id int(11), department_name varchar(45),  
created_date TIMESTAMP DEFAULT NOW());
```

2. Now import the data from following directory into departments_export table,
/user/cloudera/departments new

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Login to mysql db

```
mysql --user=retail_dba -password=cloudera  
show databases; use retail_db; show tables;
```

step 2 : Create a table as given in problem statement.

```
CREATE table departments_export (departmentid int(11), department_name varchar(45),  
created_date TIMESTAMP DEFAULT NOW()); show tables;
```

Step 3 : Export data from /user/cloudera/departmentsnew to new table departments_export sqoop

```
export -connect jdbc:mysql://quickstart:3306/retail_db \  
-username retaildba \  
--password cloudera \  
--table departments_export \  
-export-dir /user/cloudera/departments_new \  
-batch
```

Step 4 : Now check the export is correctly done or not. mysql -user*retail_dba -
password=cloudera

```
show databases; use retail_db;
```

```
show tables;
```

```
select* from departments_export;
```

Answer: A

Explanation:

rrrect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Login to mysql db

```
mysql --user=retail_dba -password=cloudera
```

```
show databases; use retail_db; show tables;
```

step 2 : Create a table as given in problem statement.

```
CREATE table departments_export (departmentid int(11), department_name varchar(45),  
created_date TIMESTAMP DEFAULT NOW()); show tables;
```

Step 3 : Export data from /user/cloudera/departmentsnew to new table departments_export sqoop

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

```
-username retaildba \
```

```
--password cloudera \
```

```
--table departments_export \
```

```
-export-dir /user/cloudera/departments_new \
```

```
-batch
```

Step 4 : Now check the export is correctly done or not. mysql -user*retail_dba - password=cloudera

```
show databases; use retail_db;
```

```
show tables;
```

```
select* from departments_export;
```

Question 6

CORRECT TEXT

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=false -
```

```
-conf XXX hadoopexam.jar
```

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution

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

Notes: ./bin/spark-submit \

```
--class <main-class>
```

```
--master <master-url> \  
--deploy-mode <deploy-mode> \  
-conf <key>=<value> \  
# other options  
< application-jar> \  
[application-arguments]
```

Here, conf is used to pass the Spark related configs 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.

Answer: A

Explanation:

rrrect.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution

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

Notes: ./bin/spark-submit \

```
--class <main-class>  
--master <master-url> \  
--deploy-mode <deploy-mode> \  
-conf <key>=<value> \  
# other options  
< application-jar> \  
[application-arguments]
```

Here, conf is used to pass the Spark related configs 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.

Question 7

CORRECT TEXT

Problem Scenario 35 : You have been given a file named spark7/EmployeeName.csv
(id,name).

EmployeeName.csv

E01,Lokesh
E02,Bhupesh
E03,Amit
E04,Ratan
E05,Dinesh
E06,Pavan
E07,Tejas
E08,Sheela
E09,Kumar
E10,Venkat

1. Load this file from hdfs and sort it by name and save it back as (id,name) in results directory. However, make sure while saving it should be able to write in a single file.

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

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 EmployeeName.csv file from hdfs and create PairRDDs

```
val name = sc.textFile("spark7/EmployeeName.csv")  
val namePairRDD = name.map(x=> (x.split(",")(0),x.split(",")(1)))
```

Step 3 : Now swap namePairRDD RDD.

```
val swapped = namePairRDD.map(item => item.swap)
```

step 4: Now sort the rdd by key.

```
val sortedOutput = swapped.sortByKey()
```

Step 5 : Now swap the result back

```
val swappedBack = sortedOutput.map(item => item.swap)
```

Step 6 : Save the output as a Text file and output must be written in a single file.

```
swappedBack.repartition(1).saveAsTextFile("spark7/result.txt")
```

Answer: A

Explanation:

rrrect.

See the explanation for Step by Step Solution and configuration.

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 EmployeeName.csv file from hdfs and create PairRDDs

```
val name = sc.textFile("spark7/EmployeeName.csv")
val namePairRDD = name.map(x=> (x.split(",")(0),x.split(",")(1)))
```

Step 3 : Now swap namePairRDD RDD.

```
val swapped = namePairRDD.map(item => item.swap)
```

step 4: Now sort the rdd by key.

```
val sortedOutput = swapped.sortByKey()
```

Step 5 : Now swap the result back

```
val swappedBack = sortedOutput.map(item => item.swap)
```

Step 6 : Save the output as a Text file and output must be written in a single file.

```
swappedBack.repartition(1).saveAsTextFile("spark7/result.txt")
```

Question 8

CORRECT TEXT

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

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

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 = new org.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.textFile("sparksql3/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(",")).map(p => Patient(p(0).toInt,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.show()
//Find all the patients who born in 2011
```

```
val results = sqlContext.sql(".....SELECT * FROM patients WHERE
YEAR(TO_DATE(CAST(UNIX_TIMESTAMP(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 TIMESTAMP))) > 60.....);
```

results. showQ;

-- Select patients 18 years old or younger

```
SELECT' FROM patients WHERE TO_DATE(CAST(UNIX_TIMESTAMP(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();

Answer: A

Explanation:

rrect.

See the explanation for Step by Step Solution and configuration.

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 = new org.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.textFile('sparksql3/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).toInt,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(UNIX_TIMESTAMP(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 TIMESTAMP))) > 60.....);
results. showQ;
-- Select patients 18 years old or younger
SELECT' FROM patients WHERE TO_DATE(CAST(UNIX_TIMESTAMP(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 9

CORRECT TEXT

Problem Scenario 46 : You have been given below list in scala (name,sex,cost) for each work done.

```
List( ("Deepak" , "male", 4000), ("Deepak" , "male", 2000), ("Deepika" , "female",
2000),("Deepak" , "female", 2000), ("Deepak" , "male", 1000) , ("Neeta" , "female", 2000))
```

Now write a Spark program to load this list as an RDD and do the sum of cost for combination of name and sex (as key)

Options:

A. Option is correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Create an RDD out of this list

```
val rdd = sc.parallelize(List( ("Deepak" , "male", 4000}, {"Deepak" , "male", 2000},
{"Deepika" , "female", 2000}, {"Deepak" , "female", 2000}, {"Deepak" , "male", 1000} ,
{"Neeta" , "female", 2000}}})
```

Step 2 : Convert this RDD in pair RDD

```
val byKey = rdd.map({case (name,sex,cost) => (name,sex)->cost})
```

Step 3 : Now group by Key

```
val byKeyGrouped = byKey.groupByKey
```

Step 4 : Now sum the cost for each group

```
val result = byKeyGrouped.map{case ((id1,id2),values) => (id1,id2,values.sum)}
```

Step 5 : Save the results result.repartition(1).saveAsTextFile("spark12/result.txt")

Answer: A

Explanation:

Correct.

See the explanation for Step by Step Solution and configuration.

Explanation:

Solution :

Step 1 : Create an RDD out of this list

```
val rdd = sc.parallelize(List( ("Deepak" , "male", 4000}, ("Deepak" , "male", 2000),  
("Deepika" , "female", 2000), ("Deepak" , "female", 2000), ("Deepak" , "male", 1000} ,  
("Neeta" , "female", 2000}})
```

Step 2 : Convert this RDD in pair RDD

```
val byKey = rdd.map({case (name,sex,cost) => (name,sex)->cost})
```

Step 3 : Now group by Key

```
val byKeyGrouped = byKey.groupByKey
```

Step 4 : Now sum the cost for each group

```
val result = byKeyGrouped.map{case ((id1,id2),values) => (id1,id2,values.sum)}
```

Step 5 : Save the results `result.repartition(1).saveAsTextFile("spark12/result.txt")`

Question 10

CORRECT TEXT

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_price | 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()`

Options:

A. Option is correct.

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]))
```

Answer: A

Explanation:

rrrect.

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-000000

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]))
```

Would you like to see more? Don't miss our CCA175 PDF file at:

<https://www.certification-questions.com/cloudera-pdf/cca175-pdf.html>