Free Questions for CCA175 by certsinside

Shared by Maxwell on 29-01-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

Problem Scenario 74 : 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 order table : (orderjd , order_date , ordercustomerid, order status}

Columns of orderjtems table : (order_item_td , order_item_order_id , order_item_product_id, order_item_quantity,order_item_subtotal,order_item_product_price)

Please accomplish following activities.

1. Copy "retaildb.orders" and "retaildb.orderjtems" table to hdfs in respective directory p89_orders and p89_order_items .

2. Join these data using orderjd in Spark and Python

3. Now fetch selected columns from joined data Orderld, Order date and amount collected on this order.

4. Calculate total order placed for each date, and produced the output sorted by date.

Options:

A- Solution:

Step 1 : Import Single table .

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -table=orders --target-dir=p89_orders - -m1

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -table=order_items ~target-dir=p89_order items -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 : Read the data from one of the partition, created using above command, hadoopfs -cat p89_orders/part-m-00000 hadoop fs -cat p89_order_items/part-m-00000

Step 3 : Load these above two directory as RDD using Spark and Python (Open pyspark terminal and do following). orders = sc.textFile('p89_orders') orderitems = sc.textFile('p89_order_items')

Step 4 : Convert RDD into key value as (orderjd as a key and rest of the values as a value)

#First value is orderjd

ordersKeyValue = orders.map(lambda line: (int(line.split(',')[0]), line))

#Second value as an Orderjd

orderItemsKeyValue = orderItems.map(lambda line: (int(line.split(',')[1]), line))

Step 5 : Join both the RDD using orderjd

joinedData = orderItemsKeyValue.join(ordersKeyValue)

#print the joined data
tor line in joinedData.collect():
print(line)
Format of joinedData as below.
[Orderld, 'All columns from orderltemsKeyValue', 'All columns from orders Key Value']
Step 6 : Now fetch selected values Orderld, Order date and amount collected on this order.
revenuePerOrderPerDay = joinedData.map(lambda row: (row[0](row[1][1].split(',')[1](f!oat(row[1][0].split('\M}[4]}}}
#printthe result
for line in revenuePerOrderPerDay.collect():
print(line)
Step 7 : Select distinct order ids for each date.
#distinct(date,order_id)
distinctOrdersDate = joinedData.map(lambda row: row[1][1].split('\')[1] + ',' + str(row[0])).distinct()
for line in distinctOrdersDate.collect(): print(line)
Step 8 : Similar to word count, generate (date, 1) record for each row. newLineTuple = distinctOrdersDate.map(lambda line: (line.split(',')[0], 1))
Step 9 : Do the count for each key(date), to get total order per date. totalOrdersPerDate = newLineTuple.reduceByKey(lambda a, b: a + b}
#print results
for line in totalOrdersPerDate.collect():
print(line)
step 10 : Sort the results by date sortedData=totalOrdersPerDate.sortByKey().collect()
#print results
for line in sortedData:
print(line)

B- Solution:

Step 1 : Import Single table .

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -table=orders --target-dir=p89_orders - -m1

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -table=order_items ~target-dir=p89_order items -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 : Read the data from one of the partition, created using above command, hadoopfs -cat p89_orders/part-m-00000 hadoop fs -cat p89_order_items/part-m-00000

Step 3 : Load these above two directory as RDD using Spark and Python (Open pyspark terminal and do following). orders = sc.textFile('p89_orders') orderitems = sc.textFile('p89_order_items')

Step 4 : Convert RDD into key value as (orderjd as a key and rest of the values as a value)

#First value is orderjd

Step 5 : Join both the RDD using orderjd

joinedData = orderItemsKeyValue.join(ordersKeyValue)

#print the joined data

tor line in joinedData.collect():

print(line)

Format of joinedData as below.

[Orderld, 'All columns from orderItemsKeyValue', 'All columns from orders Key Value']

Step 6 : Now fetch selected values Orderld, Order date and amount collected on this order.

revenuePerOrderPerDay = joinedData.map(lambda row: (row[0](row[1][1].split(',')[1](f!oat(row[1][0].split('\M}[4]}}}

Step 7 : Select distinct order ids for each date.

#distinct(date,order_id)

distinctOrdersDate = joinedData.map(lambda row: row[1][1].split('\')[1] + ',' + str(row[0])).distinct()

for line in distinctOrdersDate.collect(): print(line)
Step 8 : Similar to word count, generate (date, 1) record for each row. newLineTuple = distinctOrdersDate.map(lambda line: (line.split(',')[0], 1))
Step 9 : Do the count for each key(date), to get total order per date. totalOrdersPerDate = newLineTuple.reduceByKey(lambda a, b: a + b}
#print results
for line in totalOrdersPerDate.collect():
print(line)
step 10 : Sort the results by date sortedData=totalOrdersPerDate.sortByKey().collect()
#print results
for line in sortedData:
print(line)

Answer:

A

Question 2

Question Type: **MultipleChoice**

Problem Scenario 73 : 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 file locally.

2. Load this file 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.

Options:

A- 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

Step 3 : Write spark script
#Import SQLContext
from pyspark import SQLContext
#Create instance of SQLContext sqIContext = SQLContext(sc)
#Load json file
employee = sqlContext.jsonFile('employee.json')
#Register RDD as a temp table employee.registerTempTablef'EmployeeTab'}
#Select data from Employee table
employeeInfo = sqlContext.sql('select * from EmployeeTab'}
#Iterate data and print
for row in employeeInfo.collect():
print(row)
Step 4 : Write dataas a Text file employeeInfo.toJSON().saveAsTextFile('employeeJson1')
Step 5: Check whether data has been created or not hadoop fs -cat employeeJsonl/part'

B- 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
Step 3 : Write spark script
#Import SQLContext
employee = sqlContext.jsonFile('employee.json')
#Register RDD as a temp table employee.registerTempTablef'EmployeeTab'}
#Select data from Employee table
employeeInfo = sqlContext.sql('select * from EmployeeTab'}
#Iterate data and print

for row in employeeInfo.collect():
print(row)
Step 4 : Write dataas a Text file employeeInfo.toJSON().saveAsTextFile('employeeJson1')
Step 5: Check whether data has been created or not hadoop fs -cat employeeJsonl/part'

Answer:

A

Question 3

Question Type: **MultipleChoice**

Problem Scenario 72 : You have been given a table named "employee2" with following detail.

first_name string

last_name string

Write a spark script in python which read this table and print all the rows and individual column values.

Options:

A- Solution :

Step 1 : Import statements for HiveContext from pyspark.sql import HiveContext

Step 2 : Create sqIContext sqIContext = HiveContext(sc)

Step 3 : Query hive

employee2 = sqlContext.sql('select' from employee2')

Step 4 : Now prints the data for row in employee2.collect(): print(row)

Step 5 : Print specific column for row in employee2.collect(): print(row.fi rst_name)

B- Solution :

Step 1 : Import statements for HiveContext from pyspark.sql import HiveContext

Step 2 : Create sqIContext sqIContext = HiveContext(sc)

Step 3 : Now prints the data for row in employee2.collect(): print(row)

Step 4 : Print specific column for row in employee2.collect(): print(row.fi rst_name)

Answer:

A

Question 4

Question Type: **MultipleChoice**

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

Options:

A- 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)

B- 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 : Reading back the sequence file data using spark. seqRDD = sc.sequenceFile('problem86_1')
Step 6 : Print the content to validate the same.
for line in seqRDD.collect():
print(line)

Answer:

A

Question 5

Question Type: **MultipleChoice**

Problem Scenario 70 : Write down a Spark Application using Python, In which it read a file "Content.txt" (On hdfs) with following content. Do the word count and save the results in a directory called "problem85" (On hdfs)

Content.txt

Hello this is ABCTECH.com

This is XYZTECH.com

Apache Spark Training

This is Spark Learning Session

Spark is faster than MapReduce

Options:

A- Solution :

Step 1 : Create an application with following code and store it in problem84.py

```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf
# Create configuration object and set App name
conf = SparkConf().setAppName('CCA 175 Problem 85') sc = sparkContext(conf=conf)
#load data from hdfs
.reduceByKey(lambda x, y: x+y) \
.map(lambda x: (x[1], x[0]}}.sortByKey(False}
for word in wordcounts.collect(): print(word)
#Save final data ' wordcounts.saveAsTextFile('problem85')
```

step 2 : Submit this application

```
spark-submit -master yarn problem85.py
```

B- Solution :

Step 1 : Create an application with following code and store it in problem84.py

```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf
# Create configuration object and set App name
conf = SparkConf().setAppName('CCA 175 Problem 85') sc = sparkContext(conf=conf)
#load data from hdfs
contentRDD = sc.textFile(MContent.txt')
#filter out non-empty lines
nonemptyjines = contentRDD.filter(lambda x: len(x) > 0)
#Split line based on space
words = nonempty_lines.ffatMap(lambda x: x.split('"}}
#Do the word count
wordcounts = words.map(lambda x: (x, 1)) \
.reduceByKey(lambda x, y: x+y) \
.map(lambda x: (x[1], x[0]}}.sortByKey(False}
for word in wordcounts.collect(): print(word)
#Save final data ' wordcounts.saveAsTextFile('problem85')
```

step 2 : Submit this application

```
spark-submit -master yarn problem85.py
```

Answer:

B

Question 6

Question Type: MultipleChoice

Problem Scenario 69 : Write down a Spark Application using Python,

In which it read a file "Content.txt" (On hdfs) with following content.

And filter out the word which is less than 2 characters and ignore all empty lines.

Once doen store the filtered data in a directory called "problem84" (On hdfs)

Content.txt

Hello this is ABCTECH.com

This is ABYTECH.com

Apache Spark Training

This is Spark Learning Session

Spark is faster than MapReduce

Options:

A- Solution :

Step 1 : Create an application with following code and store it in problem84.py

```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf
# Create configuration object and set App name
conf = SparkConf().setAppName('CCA 175 Problem 84') sc = sparkContext(conf=conf)
#load data from hdfs
contentRDD = sc.textFile(MContent.txt')
#filter out non-empty lines
nonemptyjines = contentRDD.filter(lambda x: len(x) > 0)
#Split line based on space
words = nonempty_lines.ffatMap(lambda x: x.split('}}
#filter out all 2 letter words
finalRDD = words.filter(lambda x: len(x) > 2)
for word in finalRDD.collect():
print(word)
#Save final data finalRDD.saveAsTextFile('problem84M)
```

step 2 : Submit this application

```
spark-submit -master yarn problem84.py
```

B- Solution :

Step 1 : Create an application with following code and store it in problem84.py

```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf
# Create configuration object and set App name
conf = SparkConf().setAppName('CCA 175 Problem 84') sc = sparkContext(conf=conf)
#load data from hdfs
```

print(word)
#Save final data finalRDD.saveAsTextFile('problem84M)
step 2 : Submit this application
spark-submit -master yarn problem84.py

Answer:

A

Question 7

Question Type: **MultipleChoice**

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 words in each sentence, and then try to find the most frequently occuring ones.

Options:

A- 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 sortthem 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)
```

B- 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 : Finally we can apply the same reduceByKey and sort steps that we used in the wordcount example, to count up the bigrams and sortthem 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)
```

Answer:

A

Question 8

Question Type: **FillInTheBlank**

Problem Scenario 67 : You have been given below code snippet.

```
lines = sc.parallelize(['lts fun to have fun,','but you have to know how.'])

M = lines.map( lambda x: x.replace(',7 ').replace('.',' 'J.replaceC-V ').lower())

r2 = r1.flatMap(lambda x: x.split())

r3 = r2.map(lambda x: (x, 1))

operation1

r5 = r4.map(lambda x:(x[1],x[0]))
```

r6 = r5.sortByKey(ascending=False)

r6.take(20)

Write a correct code snippet for operationl which will produce desired output, shown below. [(2, 'fun'), (2, 'to'), (2, 'have'), (1, its'), (1, 'know1), (1, 'how1), (1, 'you'), (1, 'but')]

Answer:

Question 9

Question Type: **MultipleChoice**

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

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

val b = a.keyBy(_.length)

val c = sc.parallelize(List("ant", "falcon", "squid"), 2)

val d = c.keyBy(.length)

operation 1

Write a correct code snippet for operationl which will produce desired output, shown below. Array[(lnt, String)] = Array((4,lion))

Options:

A- Solution :

b.subtractByKey(d).collect

subtractByKey [Single] : Very similar to subtract, but instead of supplying a function, the key-component of each pair will be automatically used ascriterion for removing items from the first RDD.

B- Solution :

b.subtractByKey(d).collect

subtractByKey [Pair] : Very similar to subtract, but instead of supplying a function, the key-component of each pair will be automatically used ascriterion for removing items from the first RDD.

Answer:

B

Question 10

Question Type: **MultipleChoice**

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))

Options:

A- 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 datathat is output by a reducerwhich has been shuffled. The implementation of this function is actually very clever. Then it sorts these ranges individually with mapPartitions using standard sort mechanisms.

B- 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 datathat is output by a reducerwhich has been shuffled. The implementation of this function is actually very clever. First, it uses a range partitioner to partition the data in rangeswithin the shuffled RDD.
Then it sorts these ranges individually with mapPartitions using standard sort mechanisms.

Answer:

B

Question 11

Question Type: MultipleChoice

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 operationl 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)))

Options:

A- 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.

B- solution : b.rightOuterJqin(d).collect

rightOuterJoin [Pair] : Performs an right outer join using two key-value RDD.Please note that the keys must be generally comparable to make this work correctly.

Answer:

A