Free Questions for CCA175 by ebraindumps

Shared by Rivers on 12-12-2023

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

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

Options:

A- 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).

The port must be whichever one your is configured to use, which is 5050 by default.Or, for a Mesoscluster 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 onthe HADOOP CONF DIR or YARN CONF DIR variable.

B- 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 Mesoscluster 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 onthe HADOOP CONF DIR or YARN CONF DIR variable.

Answer:

B

Question 2

Question Type: **MultipleChoice**

Problem Scenario 91 : 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 tile locally.

2. Load this tile 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

val employee = sqlContext.read.json('/user/cloudera/employee.json')
employee.write.parquet('employee. parquet')
val parq_data = sqlContext.read.parquet('employee.parquet')
import org.apache.spark.sql.SaveMode prdDF.write..format('orc').saveAsTable('product ore table'}
//Change the codec.
sqlContext.setConf('spark.sql.parquet.compression.codec','snappy')
employee.write.mode(SaveMode.Overwrite).parquet('employee.parquet')

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
val employee = sqlContext.read.json('/user/cloudera/employee.json')
employee.write.parquet('employee. parquet')
val parq_data = sqlContext.read.parquet('employee.parquet')
parq_data.registerTempTable('employee')
val allemployee = sqlContext.sql('SELeCT' FROM employee')
all_employee.show()
import org.apache.spark.sql.SaveMode prdDF.write..format('orc').saveAsTable('product ore table'}
//Change the codec.
sqlContext.setConf('spark.sql.parquet.compression.codec','snappy')
employee.write.mode(SaveMode.Overwrite).parquet('employee.parquet')

Answer:

B

Question 3

Question Type: **MultipleChoice**

Problem Scenario 90 : You have been given below two files

course.txt

id,course

1,Hadoop

2,Spark

3,HBase

fee.txt

id,fee

2,3900

3,4200

4,2900

Accomplish the following activities.

1. Select all the courses and their fees , whether fee is listed or not.

2. Select all the available fees and respective course. If course does not exists still list the fee

3. Select all the courses and their fees , whether fee is listed or not. However, ignore records having fee as null.

Options:

A- Solution :

Step 1:

```
hdfs dfs -mkdir sparksql4
hdfs dfs -put course.txt sparksql4/
hdfs dfs -put fee.txt sparksql4/
```

Step 2 : Now in spark shell

```
// load the data into a new RDD
val course = sc.textFile('sparksql4/course.txt')
val fee = sc.textFile('sparksql4/fee.txt')
// Return the first element in this RDD
course.fi rst()
fee.fi rst()
//define the schema using a case class case class Course(id: Integer, name: String) case class Fee(id: Integer, fee: Integer)
// create an RDD of Product objects
val courseRDD = course.map(_.split(',')).map(c => Course(c(0).tolnt,c(1)))
val feeRDD =fee.map(_.split(',')).map(c => Fee(c(0}.tolnt,c(1}.tolnt))
```

courseRDD.first()
courseRDD.count{}
feeRDD.first()
feeRDD.countQ
// change RDD of Product objects to a DataFrame val courseDF = courseRDD.toDF{} val feeDF = feeRDD.toDF{)
// register the DataFrame as a temp table courseDF. registerTempTable('course') feeDF. registerTempTablef'fee')
// Select data from table
val results = sqlContext.sql(.....SELECT' FROM course "')
results. showQ
val results = sqlContext.sql(.....SELECT' FROM fee.....)
results. showQ
val results = sqlContext.sql(.....SELECT * FROM course LEFT JOIN fee ON course.id = fee.id.....)
results-showQ
val results ='sqlContext.sql(.....SELECT * FROM course RIGHT JOIN fee ON course.id = fee.id 'MM)
results. showQ
val results = sqlContext.sql(.....SELECT' FROM course LEFT JOIN fee ON course.id = fee.id where fee.id IS NULL'
results. show()

B- Solution :
Step 1:
hdfs dfs -mkdir sparksql4
hdfs dfs -put course.txt sparksql4/
hdfs dfs -put fee.txt sparksql4/
Step 2 : Now in spark shell
// load the data into a new RDD
val course = sc.textFile('sparksql4/course.txt')

```
val fee = sc.textFile('sparksql4/fee.txt')
// Return the first element in this RDD
course.fi rst()
fee.fi rst()
//define the schema using a case class case class Course(id: Integer, name: String) case class Fee(id: Integer, fee: Integer)
// create an RDD of Product objects
val courseRDD = course.map(_.split(',')).map(c => Course(c(0).tolnt,c(1)))
val feeRDD =fee.map(_.split(',')).map(c => Fee(c(0}.tolnt,c(1}.tolnt))
courseRDD.first()
courseRDD.count(}
feeRDD.first()
results-showQ
val results ='sqlContext.sql(.....SELECT * FROM course RIGHT JOIN fee ON course.id = fee.id 'MM )
results. showQ
val results = sqlContext.sql(.....SELECT' FROM course LEFT JOIN fee ON course.id = fee.id where fee.id IS NULL'
results. show()
```

Answer:

A

Question 4

Question Type: MultipleChoice

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.

Options:

A- 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 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()

B- 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()

Answer:

B

Question 5

Question Type: MultipleChoice

Problem Scenario 82 : You have been given table in Hive with following structure (Which you have created in previous exercise).

productid int code string name string quantity int price float

Using SparkSQL accomplish following activities.

1. Select all the products name and quantity having quantity

2. Select name and price of the product having code as 'PEN'

3. Select all the products, which name starts with PENCIL

4. Select all products which "name" begins with 'P\ followed by any two characters, followed by space, followed by zero or more characters

Options:

A- Solution :
Step 1 : Copy following tile (Mandatory Step in Cloudera QuickVM) if you have not done it.
sudo su root
cp /usr/lib/hive/conf/hive-site.xml /usr/lib/sparkVconf/
Step 2 : Now start spark-shell

Step 3 ; Select all the products name and quantity having quantity <= 2000
val results = sqlContext.sql(.....SELECT name, quantity FROM products WHERE quantity <= 2000.....)
results.showQ
Step 4 : Select name and price of the product having code as 'PEN'
val results = sqlContext.sql(.....SELECT name, price FROM products WHERE code = 'PEN.....)
results. showQ
Step 5 : Select all the products , which name starts with PENCIL
val results = sqlContext.sql(.....SELECT name, price FROM products WHERE upper(name) LIKE 'PENCIL%.....}
results. showQ
Step 6 : select all products which 'name' begins with 'P', followed by any two characters, followed by space, followed byzero or more characters
-- 'name' begins with 'P', followed by any two characters,
- followed by space, followed by zero or more characters
val results = sqlContext.sql(.....SELECT name, price FROM products WHERE name LIKE 'P_ %.....)
results. show()

B- Solution :
Step 1 : Copy following tile (Mandatory Step in Cloudera QuickVM) if you have not done it.
sudo su root
cp /usr/lib/hive/conf/hive-site.xml /usr/lib/sparkVconf/
Step 2 : Now start spark-shell
Step 3 ; Select all the products name and quantity having quantity <= 2000
val results = sqlContext.sql(.....SELECT name, quantity FROM products WHERE quantity <= 2000.....)
results.showQ
Step 4 : Select all the products , which name starts with PENCIL
val results = sqlContext.sql(.....SELECT name, price FROM products WHERE upper(name) LIKE 'PENCIL%.....}

results. showQ
Step 5 : select all products which 'name' begins with 'P', followed by any two characters, followed by space, followed byzero or more characters
-- 'name' begins with 'P', followed by any two characters,
- followed by space, followed by zero or more characters
val results = sqlContext.sql(.....SELECT name, price FROM products WHERE name LIKE 'P_ %.....)
results. show()

Answer:

A

Question 6

Question Type: **MultipleChoice**

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- 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:Integer, 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 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 7 : Now create a parquet table
import org.apache.spark.sql.SaveMode
prdDF.write.mode(SaveMode.Overwrite).format('parquet').saveAsTable('product_parquet_table')
Step 8 : 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 9 : Check data has been loaded or not.
Select * from products;
Select * from products_parquet;

B- 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:Integer, 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;

Answer:

B

Question 7

Question Type: **MultipleChoice**

Problem Scenario 80 : You have been given MySQL DB with following details.

user=retail_dba

password=cloudera

database=retail_db

table=retail_db.products

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

Columns of products table : (product_id | product_category_id | 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. Now sort the products data sorted by product price per category, use productcategoryid colunm to group by category

Options:

A- 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

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(Mp93_products')

Step 4 : Filter empty prices, if exists

#filter out empty prices lines

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

Step 5 : Create data set like (categroyld, (id,name,price)

mappedRDD = nonempty_lines.map(lambda line: (line.split(',')[1], (line.split(',')[0], line.split(',')[2], float(line.split(',')[4]))))

tor line in mappedRDD.collect(): print(line)

Step 6 : Now groupBy the all records based on categoryld, which a key on mappedRDD it will produce output like (categoryld, iterable of all linesfor a key/categoryld)

groupByCategroyld = mappedRDD.groupByKey() for line in groupByCategroyld.collect(): print(line)

step 7 : Now sort the data in each category based on price in ascending order.

sorted is a function to sort an iterable, we can also specify, what would be the Key on which we want to sort in this case we have price

onwhich it needs to be sorted.
groupByCategroyld.map(lambda tuple: sorted(tuple[1], key=lambda tupleValue: tupleValue[2])).take(5)
Step 8 : Now sort the data in each category based on price in descending order.
sorted is a function to sort an iterable, we can also specify, what would be the Key on which we want to sort in this case we have price which it needs to be sorted.
on groupByCategroyld.map(lambda tuple: sorted(tuple[1], key=lambda tupleValue: tupleValue[2] , reverse=True)).take(5)

B- 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
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(Mp93_products')
Step 4 : Filter empty prices, if exists
#filter out empty prices lines
Nonempty_lines = productsRDD.filter(lambda x: len(x.split(',')[4]) > 0)
Step 5 : Create data set like (categroyld, (id,name,price)
mappedRDD = nonempty_lines.map(lambda line: (line.split(',')[1], (line.split(',')[0], line.split(',')[2], float(line.split(',')[4]))))
tor line in mappedRDD.collect(): print(line)
step 6 : Now sort the data in each category based on price in ascending order.
# sorted is a function to sort an iterable, we can also specify, what would be the Key on which we want to sort in this case we have price onwhich it needs to be sorted.
groupByCategroyld.map(lambda tuple: sorted(tuple[1], key=lambda tupleValue: tupleValue[2])).take(5)
Step 7 : Now sort the data in each category based on price in descending order.

sorted is a function to sort an iterable, we can also specify, what would be the Key on which we want to sort in this case we have price which it needs to be sorted.
on groupByCategroyld.map(lambda tuple: sorted(tuple[1], key=lambda tupleValue: tupleValue[2] , reverse=True)).take(5)

Answer:

A