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.passapply.com/cca175.html

100% Passing Guarantee
100% Money Back Assurance

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

- **Instant Download** After Purchase
- **100% Money Back** Guarantee
- **365 Days** Free Update
- **800,000+** Satisfied Customers

QUESTION 1

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

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

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 lines for 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 on which 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)

QUESTION 2

Problem Scenario 95 : You have to run your Spark application on yarn with each executor Maximum heap size to be 512MB and Number of processor cores to allocate on each executor will be 1 and Your main application required three values as input arguments V1 V2 V3. Please replace XXX, YYY, ZZZ ./bin/spark-submit -class com.hadoopexam.MyTask --master yarn-cluster--num-executors 3 --driver-memory 512m XXX YYY lib/hadoopexam.jarZZZ

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

Solution

XXX: -executor-memory 512m YYY: -executor-cores 1 ZZZ : V1 V2 V3 Notes : spark-submit on yarn options Option Description archives Comma-separated list of archives to be extracted into the working directory of each executor. The path must be globally visible inside your cluster; see Advanced Dependency Management. executor-cores Number of processor cores to allocate on each executor. Alternatively, you can use the spark.executor.cores property, executor-memory Maximum heap size to allocate to each executor. Alternatively, you can use the spark.executor.memory-property. num-executors Total number of YARN containers to allocate for this application. Alternatively, you can use the spark.executor.instances property. queue YARN queue to submit to. For more information, see Assigning Applications and Queries to Resource Pools. Default: default.

QUESTION 3

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 TrainingThis is Spark Learning Session Spark is faster than MapReduce

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

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

QUESTION 4

Problem Scenario 3: 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.

1.

Import data from categories table, where category=22 (Data should be stored in categories subset)

2.

Import data from categories table, where category>22 (Data should be stored in categories_subset_2)

3.

Import data from categories table, where category between 1 and 22 (Data should be stored in categories_subset_3)

4.

While importing catagories data change the delimiter to \\'|\\' (Data should be stored in categories_subset_S)

5.

Importing data from catagories table and restrict the import to category_name,category id columns only with delimiter as \\'|\\'

6.

Add null values in the table using below SQL statement ALTER TABLE categories modify category_department_id int(11); INSERT INTO categories values (eO.NULL.\\'TESTING\\');

7.

Importing data from catagories table (In categories_subset_17 directory) using \\'|\\' delimiter and categoryjd between 1 and 61 and encode null values for both string and non string columns.

8.

Import entire schema retail_db in a directory categories_subset_all_tables

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

Solution: Step 1: Import Single table (Subset data} Note: Here the \\' is the same you find on - key sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail_dba password=cloudera -table=categories ~warehouse-dir= categories_subset --where \\\'category_id\'=22 --m 1 Step 2 : Check the output partition hdfs dfs -cat categoriessubset/categories/part-m-00000 Step 3 : Change the selection criteria (Subset data) sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail_dba password=cloudera -table=categories ~warehouse-dir= categories_subset_2 --where \'category_id\'\>22 -m 1 Step 4 : Check the output partition hdfs dfs -cat categories_subset_2/categories/part-m-00000 Step 5 : Use between clause (Subset data) sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail_dba password=cloudera -table=categories ~warehouse-dir=categories_subset_3 --where "\'category_id\\\' between 1 and 22" --m 1 Step 6 : Check the output partition hdfs dfs -cat categories_subset_3/categories/part-m-00000 Step 7 : Changing the delimiter during import. sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail dba password=cloudera -table=categories -warehouse-dir=:categories_subset_6 --where "/'categoryjd /' between 1 and 22" -fields-terminated-by=\\'|\\' -m 1 Step 8 : Check the.output partition hdfs dfs -cat categories_subset_6/categories/part-m-00000 Step 9 : Selecting subset columnssqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail_dba password=cloudera -table=categories --warehouse-dir=categories subset col -where "/'category id/' between 1 and 22" -fields-terminated-by=T -columns=category name,category id --m 1 Step 10 : Check the output partition hdfs dfs -cat categories_subset_col/categories/part-m-00000 Step 11 : Inserting record with null values (Using mysql} ALTER TABLE categories modify category_department_id int(11); INSERT INTO categories values ^NULL/TESTING\\'); select" from categories; Step 12 : Encode non string null column sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail dba password=cloudera -table=categories --warehouse-dir=categortes_subset_17 -where "\"category_id\" between 1 and 61" -fields-terminated-by=,|\\' --null-string-N\\' -null-nonstring=, N\\' --m 1 Step 13 : View the content hdfs dfs -cat categories_subset_17/categories/part-m-00000 Step 14 : Import all the tables from a schema (This step will take little time) sqoop import-all-tables -connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba -password=cloudera -warehouse-dir=categories_si Step 15 : View the contents

hdfs dfs -Is categories_subset_all_tables

Step 16 : Cleanup or back to originals.

delete from categories where categoryid in (59,60);

ALTER TABLE categories modify category_department_id int(11) NOTNULL;

ALTER TABLE categories modify category_name varchar(45) NOT NULL;

desc categories;

QUESTION 5

Problem Scenario 31 : You have given following two files

1.

Content.txt: Contain a huge text file containing space separated words.

2.

Remove.txt: Ignore/filter all the words given in this file (Comma Separated). Write a Spark program which reads the Content.txt file and load as an RDD, remove all the words from a broadcast variables (which is loaded as an RDD of words from Remove.txt). And count the occurrence of the each word and save it as a text file in HDFS. Content.txt Hello this is ABCTech.com This is TechABY.com Apache Spark Training This is Spark Learning Session Spark is faster than MapReduce Remove.txt Hello, is, this, the

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

Solution : Step 1 : Create all three files in hdfs in directory called spark2 (We will do using Hue).However, you can first create in local filesystem and then upload it to hdfs Step 2 : Load the Content.txt file val content = sc.textFile("spark2/Content.txt") //Load the text file Step 3 : Load the Remove.txt file val remove = sc.textFile("spark2/Remove.txt") //Load the text file Step 4 : Create an RDD from remove, However, there is a possibility each word could have trailing spaces, remove those whitespaces as well. We have used two functions here flatMap, map and trim. val removeRDD= remove.flatMap(x=> x.splitf\\',")).map(word=>word.trim)//Create an array of words Step 5 : Broadcast the variable, which you want to ignore val bRemove = sc.broadcast(removeRDD.collect().toList) // It should be array of Strings Step 6 : Split the content RDD, so we can have Array of String. val words = content.flatMap(line => line.split(" ")) Step 7 : Filter the RDD, so it can have only content which are not present in "Broadcast Variable". val filtered = words.filter{case (word) => !bRemove.value.contains(word)} Step 8 : Create a PairRDD, so we can have (word,1) tuple or PairRDD. val pairRDD = filtered.map(word => (word,1)) Step 9 : Nowdo the word count on PairRDD. val wordCount = pairRDD.reduceByKey(_ + _) Step 10 : Save the output as a Text file. wordCount.saveAsTextFile("spark2/result.txt")