PassQueen
a.k.a dumpsqueen.com

CCA Spark and Hadoop Developer Exam

Cloudera CCA175

Version Demo

Total Demo Questions: 29

Total Premium Questions: 298

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Data Ingest	77
Topic 2, Transform, Stage, and Store Data	105
Topic 3, Data Analysis	84
Topic 4, Workflow	13
Topic 5, Mix Questions	19
Total	298

QUESTION NO: 1 - (SIMULATION)

Problem Scenario 18 : 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

Now accomplish following activities.

1. Create mysql table as below.

mysql --user=retail_dba -password=cloudera

use retail_db

CREATE TABLE IF NOT EXISTS departments_hive02(id int, department_name varchar(45), avg_salary int);

show tables;

2. Now export data from hive table departments_hive01 in departments_hive02. While exporting, please note following. wherever there is a empty string it should be loaded as a null value in mysql.

wherever there is -999 value for int field, it should be created as null value.

ANSWER: See the explanation for the answer

Explanation:

Create the MySQL target table, then run a Sqoop export from the Hive managed-table directory. Use an empty input-null string so empty character fields become SQL `NULL`, and use `-999` as the null marker for non-string fields.

```
mysql -u retail_dba -pcloudera

USE retail_db;

CREATE TABLE IF NOT EXISTS departments_hive02 (
  id INT,
  department_name VARCHAR(45),
  avg_salary INT
);

SHOW TABLES;
EXIT;
```

Export the Hive table data:

```
sqoop export \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table departments_hive02 \
  --export-dir /user/hive/warehouse/departments_hive01 \
  --input-fields-terminated-by $'\001' \
  --input-lines-terminated-by $'\n' \
  --input-null-string '' \
  --input-null-non-string -999 \
  --num-mappers 1 \
  --batch
```

`--input-null-string ''` converts an empty string in `department_name` to MySQL `NULL`.
`--input-null-non-string -999` converts `-999` in integer columns such as `avg_salary` to MySQL `NULL`.
The Hive default field separator is Ctrl+A, represented by `$'\001'`.

Validate the result:

```
mysql -u retail_dba -pcloudera -e "USE retail_db; SELECT * FROM departments_hive02;"
```

QUESTION NO: 2 - (SIMULATION)

Problem Scenario 40 : You have been given sample data as below in a file called spark15/file1.txt

3070811,1963,1096,,"US","CA",,1,

3022811,1963,1096,,"US","CA",,1,56

3033811,1963,1096,,"US","CA",,1,23

Below is the code snippet to process this tile.

val field= sc.textFile("spark15/f ilel.txt")

val mapper = field.map(x=> A)

mapper.map(x => x.map(x=> {B})).collect

Please fill in A and B so it can generate below final output

Array(Array(3070811,1963,109G, 0, "US", "CA", 0,1, 0)

,Array(3022811,1963,1096, 0, "US", "CA", 0,1, 56)

,Array(3033811,1963,1096, 0, "US", "CA", 0,1, 23)

)

ANSWER: See the explanation for the answer

Explanation:

Use `split` with a negative limit so that the final empty column is retained. Replace each empty field with the integer `0`.

A: `x.split(",", -1)`
B: `if (x.isEmpty) 0 else x`

```
val field = sc.textFile("spark15/file1.txt")
val mapper = field.map(x => x.split(",", -1))
mapper.map(x => x.map(x => if (x.isEmpty) 0 else x)).collect
```

The `-1` limit is required because Scala/Java `split` otherwise discards trailing empty strings; the first input row has an empty last field that must become `0`.

QUESTION NO: 3 - (SIMULATION)

Problem Scenario 42 : You have been given a file (sparklO/sales.txt), with the content **as given in below.**

spark10/sales.txt

Department,Designation,costToCompany,State

Sales,Trainee,12000,UP

Sales,Lead,32000,AP

Sales,Lead,32000,LA

Sales,Lead,32000,TN

Sales,Lead,32000,AP

Sales,Lead,32000,TN

Sales,Lead,32000,LA

Sales,Lead,32000,LA

Marketing,Associate,18000,TN

Marketing,Associate,18000,TN

HR,Manager,58000,TN

And want to produce the output **as a csv** with group **by Department,Designation,State** with additional **columns** with sum(costToCompany) and TotalEmployeeCountt

Should get result like

Dept,Desg,state,empCount,totalCost

Sales,Lead,AP,2,64000

Sales.Lead.LA.3.96000

Sales,Lead,TN,2,64000

ANSWER: See the explanation for the answer

Explanation:

Use the Department, Designation, and trimmed State values as the grouping key. Skip the input header, then reduce each group by employee count and cost. The `trim` is important because one input row contains `TN ` with a trailing space.

```
val input = sc.textFile("spark10/sales.txt")

val grouped = input.mapPartitionsWithIndex { (idx, rows) =>
  if (idx == 0) rows.drop(1) else rows
}.map { line =>
  val c = line.split(",", -1).map(_.trim)
  ((c(0), c(1), c(3)), (1L, c(2).toLong))
}.reduceByKey { case ((count1, cost1), (count2, cost2)) =>
  (count1 + count2, cost1 + cost2)
}

val csvRows = grouped.sortByKey().map { case ((dept, desg, state), (empCount, totalCost))
=>
  s"$dept,$desg,$state,$empCount,$totalCost"
}

val output = sc.parallelize(Seq("Dept,Desg,state,empCount,totalCost"), 1)
  .union(csvRows)

output.coalesce(1).saveAsTextFile("spark10/group.txt")
```

The output directory `spark10/group.txt` will contain a CSV-style part file with:

```
Dept,Desg,state,empCount,totalCost
HR,Manager,TN,1,58000
Marketing,Associate,TN,2,36000
Sales,Lead,AP,2,64000
Sales,Lead,LA,3,96000
Sales,Lead,TN,2,64000
Sales,Trainee,UP,1,12000
```

`saveAsTextFile` writes a directory, so ensure that `spark10/group.txt` does not already exist before running the final command.

QUESTION NO: 4 - (SIMULATION)

Problem Scenario 21 : You have been given log generating service as below.

startjogs (It will generate continuous logs)

tailjogs (You can check , what logs are being generated)

stopjogs (It will stop the log service)

Path where logs are generated using above service : /opt/gen_logs/logs/access.log

Now write a flume configuration file named flumel.conf , using that configuration file dumps logs in HDFS file system in a directory called flumel. Flume channel should have following property as well. After every 100 message it should be committed, use non-durable/faster channel and it should be able to hold maximum 1000 events

Solution :

Step 1 : Create flume configuration file, with below configuration for source, sink and channel.

#Define source , sink , channel and agent,

agent1 .sources = source1

agent1 .sinks = sink1

agent1.channels = channel1

Describe/configure source1

agent1 .sources.source1.type = exec

agent1.sources.source1.command = tail -F /opt/gen logs/logs/access.log

Describe sinkl

agentl .sinks.sinkl.channel = memory-channel

agentl .sinks.sinkl .type = hdfs

agentl .sinks.sink1.hdfs.path = flumel

agentl .sinks.sinkl.hdfs.fileType = Data Stream

Now we need to define channell property.

agent1.channels.channel1.type = memory

agent1.channels.channell.capacity = 1000

agent1.channels.channell.transactionCapacity = 100

Bind the source and sink to the channel

agent1.sources.source1.channels = channel1

agent1.sinks.sink1.channel = channel1

Step 2 : Run below command which **will use this** configuration file and append data in hdfs.

Start log service using : startjogs

Start flume service:

flume-ng agent -conf /home/cloudera/flumeconf -conf-file /home/cloudera/flumeconf/flumel.conf-Dflume.root.logger=DEBUG,INFO,console

Wait for few mins and than stop log service.

Stop_logs

ANSWER: See the explanation for the answer

Explanation:

Create `/home/cloudera/flumeconf/flumel.conf` with a memory channel. The memory channel is non-durable and fast; `capacity` sets its maximum event count and `transactionCapacity` sets the 100-event commit/batch transaction size.

```
agent1.sources = source1
agent1.sinks = sink1
agent1.channels = channel1

# Source: continuously follow the generated log
agent1.sources.source1.type = exec
agent1.sources.source1.command = tail -F /opt/gen_logs/logs/access.log
agent1.sources.source1.channels = channel1

# Memory (non-durable) channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 1000
agent1.channels.channel1.transactionCapacity = 100

# HDFS sink
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.channel = channel1
agent1.sinks.sink1.hdfs.path = /user/cloudera/flumel
agent1.sinks.sink1.hdfs.fileType = DataStream
```

Create the target HDFS directory if it does not already exist:

```
hdfs dfs -mkdir -p /user/cloudera/flumel
```

Start log generation, then start the Flume agent using the configuration:

```
startjogs
flume-ng agent --conf /home/cloudera/flumeconf --conf-file
/home/cloudera/flumeconf/flumel.conf --name agent1 -Dflume.root.logger=INFO,console
```

After logs have been collected, stop the generator:

```
stopjogs
```

Verify that Flume created output files in HDFS:

```
hdfs dfs -ls /user/cloudera/flumel
hdfs dfs -cat /user/cloudera/flumel/*
```

The important corrections are to use the real log path `/opt/gen_logs/logs/access.log`, bind both source and sink to `channel1`, and use the exact sink name `sink1` consistently.

QUESTION NO: 5 - (SIMULATION)

Problem Scenario 92 : You have been given a spark scala application, which is bundled in jar named hadoopexam.jar.

Your application class name is com.hadoopexam.MyTask

You want that while submitting your application should launch a driver on one of the cluster node.

Please complete the following command to submit the application.

spark-submit XXX -master yarn \

YYY SSPARK HOME/lib/hadoopexam.jar 10

ANSWER: See the explanation for the answer

Explanation:

Use YARN `cluster` deploy mode so that the driver is launched on a cluster node, and specify the application's main class with `--class`.

```
spark-submit --class com.hadoopexam.MyTask --master yarn \
  --deploy-mode cluster $SPARK_HOME/lib/hadoopexam.jar 10

XXX = --class com.hadoopexam.MyTask
YYY = --deploy-mode cluster
```

QUESTION NO: 6 - (SIMULATION)

SIMULATION6

SIMULATION

Problem Scenario 36 : You have been given a file named spark8/data.csv (type,name).

data.csv

1,Lokesh

2,Bhupesh

2,Amit

2,Ratan

2,Dinesh

1,Pavan

1,Tejas

2,Sheela

1,Kumar

1,Venkat

1. Load this file from hdfs and save it back as (id, (all names of same type)) in results directory. However, make sure while saving it should be

ANSWER: See the explanation for the answer

Explanation:

Use a single output partition so that the `results` directory contains only one Spark `part-*` output file. In `spark-shell`, run:

```
val input = sc.textFile("spark8/data.csv")

val pairs = input
  .map(_.split(",", 2))
```

```
  .map(a => (a(0), a(1)))

val result = pairs
  .groupByKey(1)
  .map { case (id, names) =>
    s"($id,(${names.mkString(",")}))"
  }

result.coalesce(1).saveAsTextFile("spark8/results")
```

The output is grouped by the first column (type/id), for example:

```
(1,(Lokesh,Pavan,Tejas,Kumar,Venkat))
(2,(Bhupesh,Amit,Ratan,Dinesh,Sheela))
```

`groupByKey(1)` creates one result partition, and `coalesce(1)` explicitly ensures that saving creates a single output part file. If the target already exists, remove it before running the save command:

```
hdfs dfs -rm -r spark8/results
```

QUESTION NO: 7 - (SIMULATION)

Problem Scenario 10 : 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 database named hadoopexam and then create a table named departments in it, with following fields.
department_id **int,**

department_name string

e.g. location should be hdfs://quickstart.cloudera:8020/user/hive/warehouse/hadoopexam.db/departments

2. Please import data in existing table created above from retaidb.departments into hive table hadoopexam.departments.

3. Please import data in a non-existing table, means while importing create hive table named hadoopexam.departments_new

ANSWER: See the explanation for the answer

Explanation:

Create the Hive database and the pre-existing target table first:

```
hive
CREATE DATABASE hadoopexam;

CREATE TABLE hadoopexam.departments (
  department_id INT,
  department_name STRING
);

DESCRIBE FORMATTED hadoopexam.departments;
```

The managed-table location will be:

```
/user/hive/warehouse/hadoopexam.db/departments
```

Import the MySQL `retail_db.departments` data into the **existing** Hive table. The `--delete-target-dir` option removes Sqoop's temporary HDFS import directory if it is left over from an earlier attempt. Do not use `--create-hive-table` for this existing table.

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table departments \
  --hive-import \
  --hive-table hadoopexam.departments \
  --hive-overwrite \
  --delete-target-dir
```

Next, import the same source into a Hive table that does not already exist. Sqoop creates `hadoopexam.departments_new` during this import:

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table departments \
  --hive-import \
  --hive-table hadoopexam.departments_new \
  --create-hive-table \
  --delete-target-dir
```

Verify both imports:

```
hive
SELECT * FROM hadoopexam.departments;
SELECT * FROM hadoopexam.departments_new;
DESCRIBE FORMATTED hadoopexam.departments_new;
```

If `--delete-target-dir` is not used, manually remove the prior Sqoop target directory (typically `/user/cloudera/departments`) before rerunning the import:

```
hdfs dfs -rm -r -f /user/cloudera/departments
```

QUESTION NO: 8 - (SIMULATION)

Problem Scenario 38 : You have been given an RDD as below,

val rdd: RDD[Array[Byte]]

Now you have to save this RDD as a SequenceFile. And below is the code snippet.

import org.apache.hadoop.io.compress.GzipCodec

rdd.map(bytesArray => (A.get(), new B(bytesArray))).saveAsSequenceFile('7output/path",classOt[GzipCodec])

What would be the correct replacement for A **and** B in above snippet.

ANSWER: See the explanation for the answer

Explanation:

The correct replacements are:

```
import org.apache.hadoop.io.{BytesWritable, NullWritable}
import org.apache.hadoop.io.compress.GzipCodec

rdd.map(bytesArray => (NullWritable.get(), new BytesWritable(bytesArray)))
   .saveAsSequenceFile("/output/path", classOf[GzipCodec])
```

`A = NullWritable`, because the RDD has no meaningful key and each record only contains a byte array value.
`B = BytesWritable`, because Hadoop SequenceFiles require Hadoop `Writable` values, and `BytesWritable` wraps an `Array[Byte]`.

Thus the required answer is `NullWritable` and `BytesWritable`.

QUESTION NO: 9 - (SIMULATION)

SIMULATION7

SIMULATION

Problem Scenario 47 : You have been given below code snippet, with intermediate output.

val z = sc.parallelize(List(1,2,3,4,5,6), 2)

// lets first print out the contents of the RDD with partition labels

def myfunc(index: Int, iter: Iterator[(Int)]): Iterator[String] = {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

//In each run , output could be different, while solving problem assume belowm output only.

z.mapPartitionsWithIndex(myfunc).collect

res28: Array[String] = Array([partID:0, val: 1], [partID:0, val: 2], [partID:0, val: 3], [partID:1, val: 4], [partID:1, val: S], [partID:1, val: 6])

Now apply aggregate method on RDD z , with two reduce function , first will select max value in each partition and second will add all the maximum values from all partitions.

Initialize the aggregate with value 5. hence expected output will be 16.

ANSWER: See the explanation for the answer

Explanation:

Run the following Spark Scala command:

```
val result = z.aggregate(5)(
  (acc, value) => Math.max(acc, value),
  (left, right) => left + right
)

println(result)   // 16
```

The sequence operation selects the maximum value within each partition, beginning with the initial value `5`:
Partition 0 (`1,2,3`) produces `5`.
Partition 1 (`4,5,6`) produces `6`.

The combine operation adds partition results. Since `aggregate` also uses the supplied zero value when combining, the result is `5 + 5 + 6 = 16`.

QUESTION NO: 10 - (SIMULATION)

SIMULATION5

SIMULATION

Problem Scenario 45 : You have been given 2 files , with the content as given Below

(spark12/technology.txt)

(spark12/salary.txt)

(spark12/technology.txt)

first,last,technology

Amit,Jain,java

Lokesh,kumar,unix

Mithun,kale,spark

Rajni,vekat,hadoop

Rahul,Yadav,scala

(spark12/salary.txt)

first,last,salary

Amit,Jain,100000

Lokesh,kumar,95000

Mithun,kale,150000

Rajni,vekat,154000

Rahul,Yadav,120000

Write a Spark program, which will join the data based on first and last name and save the joined results in following format, first Last.technology.salary

ANSWER: See the explanation for the answer

Explanation:

Use `(first,last)` as the join key, remove the header from each input, and format the joined values before writing them. Run the following in `spark-shell` (ensure the output directory does not already exist):

```
val technology = sc.textFile("/spark12/technology.txt")
  .map(_.split(",", -1))
  .filter(a => a(0) != "first")
  .map(a => ((a(0), a(1)), a(2)))

val salary = sc.textFile("/spark12/salary.txt")
  .map(_.split(",", -1))
  .filter(a => a(0) != "first")
  .map(a => ((a(0), a(1)), a(2)))

technology.join(salary)
  .map { case ((first, last), (tech, sal)) =>
    s"$first $last.$tech.$sal"
  }
  .saveAsTextFile("/spark12/joined_output")
```

The output records will be in the requested format:

```
Amit Jain.java.100000
Lokesh kumar.unix.95000
Mithun kale.spark.150000
Rajni vekat.hadoop.154000
Rahul Yadav.scala.120000
```

`join` performs an inner join on both `first` and `last`. The final `map` is required so that Spark writes the requested text format rather than its default tuple representation.

QUESTION NO: 11 - (SIMULATION)

SIMULATION7

SIMULATION

Problem Scenario 57 : You have been given below code snippet.

val a = sc.parallelize(1 to 9, 3) operationl

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

Array[(String, Seq[lnt])] = Array((even,ArrayBuffer(2, 4, G, 8)), (odd,ArrayBuffer(1, 3, 5, 7, 9)))

ANSWER: See the explanation for the answer

Explanation:

Use `groupBy` to assign each integer to either the `even` or `odd` key, then collect the result:

```
val operation1 = a.groupBy(x => if (x % 2 == 0) "even" else "odd").collect
```

Or, if only the expression replacing `operation1` is required:

```
a.groupBy(x => if (x % 2 == 0) "even" else "odd").collect
```

This produces grouped values equivalent to:

```
Array((even, ArrayBuffer(2, 4, 6, 8)),
      (odd,  ArrayBuffer(1, 3, 5, 7, 9)))
```

QUESTION NO: 12 - (SIMULATION)

Problem Scenario 63 : You have been given below code snippet.

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

val b = a.map(x => (x.length, x))

operation1

Write a correct code snippet for operationl which will produce desired output, shown below. Array[(lnt, String}] = Array((4,lion), (3,dogcat), (7,panther), (5,tigereagle))

ANSWER: See the explanation for the answer

Explanation:

Use `reduceByKey` to concatenate the words that have the same length:

```
val result = b.reduceByKey(_ + _).collect()
```

Thus, `operation1` is:

```
b.reduceByKey(_ + _).collect()
```

This groups the tuples in `b` by their integer key (word length) and reduces the string values:

`3`: `dog` + `cat` = `dogcat`

5: `tiger` + `eagle` = `tigereagle`

The resulting collected array contains the expected entries: `(4,lion)`, `(3,dogcat)`, `(7,panther)`, and `(5,tigereagle)`. The ordering of entries returned by `collect()` should not generally be relied upon.

QUESTION NO: 13 - (SIMULATION)

Problem Scenario 92 : You have been given a spark scala application, which is bundled in jar named hadoopexam.jar.

Your application class name is com.hadoopexam.MyTask

You want that while submitting your application should launch a driver on one of the cluster node.

Please complete the following command to submit the application.

spark-submit XXX -master yarn \

YYY SSPARK HOME/lib/hadoopexam.jar 10

ANSWER: See the explanation for the answer

Explanation:

Use the application main class and YARN `cluster` deploy mode so that the driver runs on a cluster node rather than on the submission host.

```
spark-submit --class com.hadoopexam.MyTask --master yarn \
  --deploy-mode cluster $SPARK_HOME/lib/hadoopexam.jar 10
```

XXX: `--class com.hadoopexam.MyTask`
YYY: `--deploy-mode cluster`

QUESTION NO: 14 - (SIMULATION)

Problem Scenario 33 : You have given a files as below.

spark5/EmployeeName.csv (id,name)

spark5/EmployeeSalary.csv (id,salary)

Data is given below:

EmployeeName.csv

E01,Lokesh

E02,Bhupesh

E03,Amit

E04,Ratan

E05,Dinesh

E06,Pavan

E07,Tejas

E08,Sheela

E09,Kumar

E10,Venkat

EmployeeSalary.csv

E01,50000

E02,50000

E03,45000

E04,45000

E05,50000

E06,45000

E07,50000

E08,10000

E09,10000

E10,10000

Now write a Spark code in scala which will load these two tiles from hdfs and join the same, and produce the (name.salary) values.

And save the data in multiple tile group by salary (Means each file will have name of employees with same salary). Make sure file name include salary as well.

ANSWER: See the explanation for the answer

Explanation:

Use the employee ID as the join key, then make salary the key so that all names with the same salary are grouped together. The following Scala code reads the HDFS files and creates one Spark output directory per salary. Each output directory name contains its salary value.

```
val employeeName = sc.textFile("/spark5/EmployeeName.csv")
val employeeSalary = sc.textFile("/spark5/EmployeeSalary.csv")

// (employeeId, employeeName)
val names = employeeName.map { line =>
  val a = line.split(",")
  (a(0), a(1))
}

// (employeeId, salary)
val salaries = employeeSalary.map { line =>
  val a = line.split(",")
  (a(0), a(1))
}

// Join by employee ID: (id, (name, salary))
val joined = names.join(salaries)

// Make salary the key: (salary, name), then group employees by salary
val salaryGroups = joined
  .map { case (_, (name, salary)) => (salary, name) }
  .groupByKey()
  .collect()

// Write the employee names for each salary into a separate output path
salaryGroups.foreach { case (salary, employeeNames) =>
  sc.parallelize(employeeNames.toSeq)
    .saveAsTextFile("/spark5/EmployeeSalary_" + salary)
}
```

This produces output directories such as:

`/spark5/EmployeeSalary_50000` containing `Lokesh`, `Bhupesh`, `Dinesh`, and `Tejas`.
`/spark5/EmployeeSalary_45000` containing `Amit`, `Ratan`, and `Pavan`.
`/spark5/EmployeeSalary_10000` containing `Sheela`, `Kumar`, and `Venkat`.

Note: Spark writes a directory for every `saveAsTextFile` call, with one or more `part-*` files inside it. Therefore the salary is included in the output directory name. If these output paths already exist, delete them from HDFS before rerunning the code.

QUESTION NO: 15 - (SIMULATION)

SIMULATION4

SIMULATION

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

ANSWER: See the explanation for the answer

Explanation:

Run the following Spark SQL statements against the `products` table/view created in the previous simulation.

1. Select products with a null product code:

```
val results1 = sqlContext.sql("SELECT * FROM products WHERE code IS NULL")
results1.show()
```

2. Select products whose name begins with `Pen`, ordered by price descending:

```
val results2 = sqlContext.sql("SELECT * FROM products WHERE name LIKE 'Pen%' ORDER BY
price DESC")
results2.show()
```

3. Select products whose name begins with `Pen`, ordered by price descending and quantity ascending:

```
val results3 = sqlContext.sql("SELECT * FROM products WHERE name LIKE 'Pen%' ORDER BY
price DESC, quantity ASC")
results3.show()
```

4. Select the top two products by price:

```
val results4 = sqlContext.sql("SELECT * FROM products ORDER BY price DESC LIMIT 2")
results4.show()
```

Use `IS NULL`, not `= NULL`, because SQL null values cannot be tested with equality. The pattern `'Pen%'` matches every product name starting with `Pen`.

QUESTION NO: 16 - (SIMULATION)

Problem Scenario 62 : You have been given below code snippet.

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

val b = a.map(x = > (x.length, x))

operation1

Write a correct code snippet for operationl which will produce desired output, shown below. Array[(lnt, String)] = Array((3,xdogx), (5,xtigerx), (4,xlionx), (3,xcatx), (7,xpantherx), (5,xeaglex))

ANSWER: See the explanation for the answer

Explanation:

Use `mapValues` to preserve each length key and add `x` before and after each animal-name value:

```
b.mapValues(x => "x" + x + "x").collect()
```

This produces:

```
Array((3,xdogx), (5,xtigerx), (4,xlionx),
      (3,xcatx), (7,xpantherx), (5,xeaglex))
```

`mapValues` transforms only the second element of each `(length, string)` tuple, leaving the length unchanged.

QUESTION NO: 17 - (SIMULATION)

SIMULATION1

SIMULATION

Problem Scenario 51 : You have been given below code snippet.

val a = sc.parallelize(List(1, 2,1, 3), 1)

val b = a.map((_, "b"))

val c = a.map((_, "c"))

Operation_xyz

Write a correct code snippet for Operationxyz which will produce below output.

Output:

Array[(lnt, (Iterable[String], Iterable[String]))] = Array(

(2,(ArrayBuffer(b),ArrayBuffer(c))),

(3,(ArrayBuffer(b),ArrayBuffer(c))),

(1,(ArrayBuffer(b, b),ArrayBuffer(c, c)))

)

ANSWER: See the explanation for the answer

Explanation:

Use `cogroup` on the two pair RDDs:

```
b.cogroup(c).collect()
```

Equivalently, the required operation is:

```
val result = b.cogroup(c)
```

`cogroup` groups values from both RDDs by their common key. Therefore, key `1` retains both duplicate values from each RDD, producing the equivalent of `(1, (Iterable(b, b), Iterable(c, c)))`, while keys `2` and `3` each have one `b` and one `c`.

QUESTION NO: 18 - (SIMULATION)

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

ANSWER: See the explanation for the answer

Explanation:

Replace `XXX` with:

```
--master local[8]
```

Complete command:

```
spark-submit --class com.hadoopexam.MyTask --master local[8] \
  --deploy-mode cluster $SPARK_HOME/lib/hadoopexam.jar 10
```

`local[8]` runs Spark locally using 8 worker threads, typically corresponding to 8 local CPU cores.

Note: In practical Spark usage, `--deploy-mode cluster` is generally for cluster managers such as YARN and is not meaningful with `local[8]`. However, the required replacement for the simulation blank is `--master local[8]`.

QUESTION NO: 19 - (SIMULATION)

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.

ANSWER: See the explanation for the answer

Explanation:

Start `spark-shell` with Hive support enabled (in the exam environment this is normally the default). Assuming the input file is located at `/user/cloudera/he/exam/task1/product.csv`, execute the following. This version handles the displayed header row correctly.

```
import org.apache.spark.sql.types._

val schema = StructType(Seq(
  StructField("productID", IntegerType, true),
  StructField("productCode", StringType, true),
  StructField("name", StringType, true),
  StructField("quantity", IntegerType, true),
  StructField("price", DoubleType, true)
))

val products = spark.read
  .option("header", "true")
  .option("mode", "FAILFAST")
  .schema(schema)
  .csv("/user/cloudera/he/exam/task1/product.csv")

products.write.mode("overwrite").format("orc")
  .saveAsTable("product_orc_table")

products.write.mode("overwrite").format("parquet")
  .saveAsTable("product_parquet_table")
```

`saveAsTable` both writes the data in the requested ORC/Parquet format and creates the corresponding Hive metastore table; no separate `CREATE EXTERNAL TABLE` command is required.

Verify the result using Spark SQL or Hive:

```
spark.sql("SHOW TABLES").show()
spark.sql("SELECT * FROM product_orc_table").show()
spark.sql("SELECT * FROM product_parquet_table").show()
```

If the supplied file has already had its header removed, use `.option("header", "false")` instead. The required final Hive tables are `product_orc_table` stored as ORC and `product_parquet_table` stored as Parquet.

QUESTION NO: 20 - (SIMULATION)

Problem Scenario 12 : 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 retailedb with following definition.

CREATE table departments_new (department_id int(11), department_name varchar(45), created_date T1MESTAMP DEFAULT NOW());

2. Now isert records from departments table to departments_new

3. **Now** import data from departments_new table to hdfs.

4. Insert following 5 records in departmentsnew table. Insert into departments_new values(110, **"Civil" , null);** Insert into departments_new values(111, "Mechanical" , **null);** Insert into departments_new values(112, "Automobile" , **null);** Insert into departments_new values(113, "Pharma" , **null);**

Insert into departments_new values(114, "Social Engineering" , null);

5. Now do the incremental import based on created_date column.

ANSWER: See the explanation for the answer

Explanation:

Create and populate the MySQL table first. The column type is `TIMESTAMP` (not `T1MESTAMP`), and the incremental check column must be named `created_date`.

```
mysql -u retail_dba -pcloudera retail_db

CREATE TABLE departments_new (
  department_id INT(11),
  department_name VARCHAR(45),
  created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO departments_new (department_id, department_name)
SELECT department_id, department_name
FROM departments;

SELECT MAX(created_date) AS initial_last_value FROM departments_new;
```

Record the value returned by `initial_last_value`; it is the `--last-value` to use for the subsequent incremental import. For example, assume it returns `2020-01-01 10:00:00`.

Perform the initial Sqoop import. Using one mapper avoids requiring a split column; alternatively, `--split-by department_id` can be used with multiple mappers.

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table departments_new \
  --target-dir /user/cloudera/departments_new \
  --num-mappers 1
```

Insert the five new rows. In MySQL, explicitly supplying `NULL` for a non-null TIMESTAMP column with a current-timestamp default results in the current timestamp in the usual QuickStart MySQL configuration. Omitting `created_date` is also a reliable way to apply the default.

```
INSERT INTO departments_new VALUES (110, 'Civil', NULL);
INSERT INTO departments_new VALUES (111, 'Mechanical', NULL);
INSERT INTO departments_new VALUES (112, 'Automobile', NULL);
INSERT INTO departments_new VALUES (113, 'Pharma', NULL);
INSERT INTO departments_new VALUES (114, 'Social Engineering', NULL);
COMMIT;
```

Finally, append only rows modified after the timestamp captured before these five inserts. Replace the example timestamp below with the actual result from the earlier `MAX(created_date)` query.

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table departments_new \
  --target-dir /user/cloudera/departments_new \
  --append \
  --incremental lastmodified \
```

```
  --check-column created_date \
  --last-value '2020-01-01 10:00:00' \
  --num-mappers 1
```

The important points are that the initial import occurs before the five inserts, the incremental mode is `lastmodified`, the check column is `created_date`, and the incremental target directory is the same directory with `--append`. Do not use `--split-by departments`, because `departments` is a table name, not a column; use `department_id` if a split column is needed.

QUESTION NO: 21 - (SIMULATION)

Problem Scenario 44 : You have been given 4 files , with the content as given below:

spark11/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

spark11/file2.txt

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.

spark11/file3.txt

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

spark11/file4.txt

Apache Storm is focused on stream processing or what some call complex event processing. Storm implements a fault tolerant method for performing a computation or pipelining multiple computations on an event as it flows into a system. One might use Storm to transform unstructured data as it flows into a system into a desired format

(spark11Afile1.txt)

(spark11/file2.txt)

(spark11/file3.txt)

(sparkl 1/file4.txt)

Write a Spark program, which will give you the highest occurring words in each file. With their file name and highest occurring words.

ANSWER: See the explanation for the answer

Explanation:

Use `wholeTextFiles` so that each input record contains both the file path and the complete contents of that file. Count normalized words using a composite key of `(fileName, word)`, find the maximum count for each file, and retain every word tied at that maximum.

Run the following in `spark-shell` (assuming the four input files are in HDFS under `spark11/`):

```
val input = sc.wholeTextFiles("spark11/file*.txt")

val wordCounts = input.flatMap { case (path, text) =>
  val fileName = path.split("/").last
  text.toLowerCase
    .split("[^\\p{L}\\p{Nd}]+")
```

```
    .filter(_.nonEmpty)
    .map(word => ((fileName, word), 1L))
}.reduceByKey(_ + _)

val maximumByFile = wordCounts
  .map { case ((fileName, word), count) => (fileName, count) }
  .reduceByKey((left, right) => math.max(left, right))

val highestWords = wordCounts
  .map { case ((fileName, word), count) => (fileName, (word, count)) }
  .join(maximumByFile)
  .filter { case (fileName, ((word, count), maximum)) => count == maximum }
  .map { case (fileName, ((word, count), maximum)) =>
    fileName + " -> " + word + " - " + count
  }
  .sortBy(line => line)

highestWords.saveAsTextFile("spark11/highest_words")
```

The output directory is `spark11/highest_words`. The program converts words to lowercase and removes punctuation, so values such as `Hadoop` and `Hadoop,` are counted as the same word.

For the supplied text, the highest-count results include:

`file1.txt`: count `2`; there are multiple tied words, including `and`, `are`, `distributed`, `framework`, `hadoop`, `hardware`, `in`, and `the`.
`file2.txt -> hadoop - 4`
`file3.txt -> data - 3` and `file3.txt -> to - 3`
`file4.txt -> a - 5`

Keeping ties is important because the requirement says “highest occurring words”; selecting only one arbitrary word for a file would discard valid highest-frequency results.

QUESTION NO: 22 - (SIMULATION)

SIMULATION

Problem Scenario 6 : 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

Compression Codec : org.apache.hadoop.io.compress.SnappyCodec

Please accomplish following.

1. Import entire database such that it can be used as a hive tables, it must be created in default schema.

2. Also make sure each tables file is partitioned in 3 files e.g. part-00000, part-00002, part-00003

3. Store all the Java files in a directory called java_output to evalute the further

ANSWER: See the explanation for the answer

Explanation:

Run the Sqoop import from a writable working directory. The `-m 3` option creates three mapper output files for every imported table, and `--outdir java_output` stores Sqoop-generated Java source files in the required directory.

```
sqoop import-all-tables \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  -m 3 \
  --hive-import \
  --create-hive-table \
  --hive-overwrite \
  --compress \
  --compression-codec org.apache.hadoop.io.compress.SnappyCodec \
  --outdir java_output
```

This imports every table from `retail_db` as a Hive table in the default Hive database/schema. The imported table files are Snappy-compressed and, with three mappers, each table should contain three data part files (for example `part-m-00000`, `part-m-00001`, and `part-m-00002`; exact prefixes can vary by Hadoop version).

If tables from an earlier attempt already exist, first drop them from the Hive `default` database (or otherwise clean their data) before running the command, so that `--create-hive-table` can create the requested tables cleanly.

Useful verification commands:

```
hive -e 'USE default; SHOW TABLES;'
hdfs dfs -ls /user/hive/warehouse/customers
ls -l java_output
```

QUESTION NO: 23 - (SIMULATION)

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[(Int, String)] = Array((4,lion))

ANSWER: See the explanation for the answer

Explanation:

Use `subtractByKey` to remove all key/value pairs in `b` whose keys are present in `d`:

```
val result = b.subtractByKey(d).collect()
```

This produces:

```
Array((4,lion))
```

`b` is keyed by string length. The keys in `d` are 3, 6, and 5 (for `ant`, `falcon`, and `squid`). Removing those keys from `b` leaves only `lion`, whose key is 4.

QUESTION NO: 24 - (SIMULATION)

SIMULATION

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

ANSWER: See the explanation for the answer

Explanation:

The relevant column is `category_id` (not `category` or `categoryjd`). Run the following Sqoop commands from the terminal. A warehouse directory is used so that Sqoop creates a `categories` subdirectory beneath each specified directory.

```
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

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

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
```

Import the categories data with the pipe field delimiter into `categories_subset_S`:

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
```

```
  --username retail_dba --password cloudera \
  --table categories \
  --warehouse-dir categories_subset_S \
  --fields-terminated-by '|' \
  --m 1
```

For the columns-only import, use `--columns`. The question does not state an output directory for this item; the following uses the conventional `categories_subset_col` directory:

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba --password cloudera \
  --table categories \
  --warehouse-dir categories_subset_col \
  --columns category_name,category_id \
  --fields-terminated-by '|' \
  --m 1
```

Create a nullable department ID column and insert a test row in MySQL. The intended malformed insert in the question is an insert such as category ID 60, a null department ID, and name `TESTING`:

```
mysql -u retail_dba -pcloudera retail_db

ALTER TABLE categories MODIFY category_department_id INT(11) NULL;
INSERT INTO categories (category_id, category_department_id, category_name)
VALUES (60, NULL, 'TESTING');
```

Import IDs 1 through 61 with a pipe delimiter and explicit null encodings for strings and non-strings:

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba --password cloudera \
  --table categories \
  --warehouse-dir categories_subset_17 \
  --where 'category_id BETWEEN 1 AND 61' \
  --fields-terminated-by '|' \
  --null-string 'NULL' \
  --null-non-string 'NULL' \
  --m 1
```

Finally, import every table in the `retail_db` schema:

```
sqoop import-all-tables \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba --password cloudera \
  --warehouse-dir categories_subset_all_tables \
  --m 1
```

For example, verify an individual table import with `hdfs dfs -cat categories_subset_17/categories/part-m-00000`, and verify the schema import with `hdfs dfs -ls categories_subset_all_tables`.

QUESTION NO: 25 - (SIMULATION)

SIMULATION0

SIMULATION

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

ANSWER: See the explanation for the answer

Explanation:

Import the MySQL `products` table into the required HDFS directory, `p93_products`:

```
sqoop import \
  --connect jdbc:mysql://quickstart:3306/retail_db \
  --username retail_dba \
  --password cloudera \
  --table products \
  --target-dir /user/cloudera/p93_products \
  --fields-terminated-by '\t' \
  --num-mappers 1
```

In `pyspark`, load the imported files, remove rows without a price, group records by `product_category_id` (field 2), and sort each category's products by `product_price` (field 5):

```
products = sc.textFile("/user/cloudera/p93_products")

# (category_id, (product_id, product_name, product_price))
byCategory = (products
    .map(lambda x: x.split("\t"))
    .filter(lambda p: len(p) > 4 and p[4] != "")
    .map(lambda p: (p[1], (p[0], p[2], float(p[4])))))

# Result: (category_id, [(product_id, product_name, price), ...])
# Products in every category are in ascending price order.
sortedByPricePerCategory = byCategory.groupByKey().mapValues(
    lambda products: sorted(products, key=lambda product: product[2])
)

sortedByPricePerCategory.take(5)
```

If the required order is descending instead, use:

```
sortedByPricePerCategory = byCategory.groupByKey().mapValues(
    lambda products: sorted(products, key=lambda product: product[2], reverse=True)
)
```

The important required import destination is `p93_products`; using `--fields-terminated-by '\t'` avoids problems if a product description or name contains commas.

QUESTION NO: 26 - (SIMULATION)

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 <= 2000

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

ANSWER: See the explanation for the answer

Explanation:

In `spark-shell`, run the following Spark SQL statements (assuming the Hive table is named `products`).

```
// 1. Product name and quantity where quantity is at most 2000
val q1 = sqlContext.sql("""
  SELECT name, quantity
  FROM products
  WHERE quantity <= 2000
""")
q1.show()

// 2. Name and price for products whose code is PEN
val q2 = sqlContext.sql("""
  SELECT name, price
  FROM products
  WHERE code = 'PEN'
""")
q2.show()

// 3. All columns for products with names starting with PENCIL
val q3 = sqlContext.sql("""
  SELECT *
  FROM products
  WHERE name LIKE 'PENCIL%'
""")
q3.show()

// 4. Name starts with P, then exactly two characters, then a space,
//    then zero or more additional characters
val q4 = sqlContext.sql("""
  SELECT *
  FROM products
  WHERE name LIKE 'P__ %'
""")
q4.show()
```

`%` matches zero or more characters.
_ matches exactly one character. Therefore `'P__ %'` means: `P`, two arbitrary characters, one literal space, and then any remaining characters.

QUESTION NO: 27 - (SIMULATION)

Problem Scenario 29 : Please accomplish the following exercises using HDFS command line options.

1. Create a directory in hdfs named hdfs_commands.

2. Create a file in hdfs named data.txt in hdfs_commands.

3. Now copy this data.txt file on local filesystem, however while copying file please make sure file properties are not changed e.g. file permissions.

4. Now create a file in local directory named data_local.txt and move this file to hdfs in hdfs_commands directory.

5. Create a file data_hdfs.txt in hdfs_commands directory and copy it to local file system.

6. Create a file in **local** filesystem named file1.txt and put it to hdfs

ANSWER: See the explanation for the answer

Explanation:

Run the following commands. The HDFS path `hdfs_commands` is relative to the current HDFS user's home directory. A local working directory is used only to make the local-file locations explicit.

```
LOCALDIR=$HOME/hdfs_command_work
mkdir -p "$LOCALDIR"

# 1. Create the HDFS directory
hdfs dfs -mkdir hdfs_commands

# 2. Create an empty HDFS file
hdfs dfs -touchz hdfs_commands/data.txt

# 3. Copy the HDFS file locally and preserve properties
hdfs dfs -copyToLocal -p hdfs_commands/data.txt "$LOCALDIR/data.txt"

# 4. Create a local file and move it into HDFS
: > "$LOCALDIR/data_local.txt"
hdfs dfs -moveFromLocal "$LOCALDIR/data_local.txt" hdfs_commands/

# 5. Create another HDFS file and copy it to local storage
hdfs dfs -touchz hdfs_commands/data_hdfs.txt
hdfs dfs -get hdfs_commands/data_hdfs.txt "$LOCALDIR/data_hdfs.txt"

# 6. Create a local file and put (copy) it into HDFS
: > "$LOCALDIR/file1.txt"
hdfs dfs -put "$LOCALDIR/file1.txt" hdfs_commands/
```

`-copyToLocal -p` is required in step 3; `-p` preserves file attributes such as permissions, ownership, and timestamps where supported.
`-moveFromLocal` transfers the local `data_local.txt` to HDFS and removes the original local source after a successful move.
`-put` copies `file1.txt` to HDFS but leaves the local source file in place.

QUESTION NO: 28 - (SIMULATION)

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

List((" Deeapak " , " 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)

ANSWER: See the explanation for the answer

Explanation:

Create an RDD from the Scala list, convert each record to a pair whose key is `(name, sex)`, and use `reduceByKey` to sum the costs. `reduceByKey` is preferred to `groupByKey` because it performs aggregation efficiently.

```
val data = List(
  ("Deeapak", "male", 4000),
  ("Deepak", "male", 2000),
```

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

val rdd = sc.parallelize(data)

val result = rdd
  .map { case (name, sex, cost) => ((name, sex), cost) }
  .reduceByKey(_ + _)
  .map { case ((name, sex), totalCost) => (name, sex, totalCost) }

result.collect.foreach(println)
// Optional output:
result.repartition(1).saveAsTextFile("spark12/result.txt")
```

The aggregated output is:

```
(Deeapak,male,4000)
(Deepak,male,3000)
(Deepika,female,2000)
(Deepak,female,2000)
(Neeta,female,2000)
```

If the intended first name was `Deepak` rather than the supplied `Deeapak`, then its male total is `7000` instead, because all three male records are grouped under the same key.

QUESTION NO: 29 - (SIMULATION)

Problem Scenario 32 : You have given three files as below.

spark3/sparkdir1/file1.txt

spark3/sparkd ir2ffile2.txt

spark3/sparkd ir3Zfile3.txt

Each file contain some text.

spark3/sparkdir1/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

spark3/sparkdir2/file2.txt

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.

spark3/sparkdir3/file3.txt

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

Now write a Spark code in scala which will load all these three files from hdfs and do the word count by filtering following words. And result should be sorted by word count in reverse order.

Filter words ("a","the","an", "as", "a","with","this","these","is","are","in", "for", "to","and","The","of")

Also please make sure you load all three files as a Single RDD (All three files must be loaded using single API call).

You have also been given following codec

import org.apache.hadoop.io.compress.GzipCodec

Please use above codec to compress file, while saving in hdfs.

ANSWER: See the explanation for the answer

Explanation:

Run the following in `spark-shell`. The comma-separated input paths are supplied to one `sc.textFile` call, so all three files are loaded as one RDD.

```
import org.apache.hadoop.io.compress.GzipCodec

val input = sc.textFile(
  "spark3/sparkdir1/file1.txt," +
  "spark3/sparkdir2/file2.txt," +
  "spark3/sparkdir3/file3.txt"
)

val excluded = Set(
  "a", "the", "an", "as", "with", "this", "these", "is", "are",
  "in", "for", "to", "and", "The", "of"
)

val wordCounts = input
  .flatMap(_.split("\\s+"))
  .map(_.trim)
  .filter(word => word.nonEmpty && !excluded.contains(word))
  .map(word => (word, 1))
  .reduceByKey(_ + _)

val sortedOutput = wordCounts.sortBy({ case (_, count) => count }, ascending = false)

sortedOutput.saveAsTextFile("spark3/compressedresult", classOf[GzipCodec])
```

The output is sorted in descending order by count.
The output directory `spark3/compressedresult` must not already exist; remove it first if rerunning the command.
Using `classOf[GzipCodec]` creates gzip-compressed Spark part files in HDFS.