Practice for CCA175

For the practice purpose please download new database from here if you have not installed it.

https://github.com/manorite/Bootcamp/blob/master/SQL/mysqlsampledatabase.sql

save it on your virtual machine home or the gateway node of the cluster.

Login to mysql as root

Open terminal

mysql – u root -p cloudera

create database classic;

create user classic_dba identified by 'classic';

grant all on classic.* to classic_dba;

flush privileges;

exit;

open a new terminal

mysql -u classic_dba -p

password

classic

show databases;

use classic;

source mysqlsampledatabase.sql

We will use two datasets (retail_db and classic) for all the practice that we need for the exams (CCA175)

Transform, Stage and Store

Open terminal

Create a new directory for practice

Hadoop fs -mkdir practice

Introduction to SCALA

Scala complies to JAVA to JVM (java byte code run by JVM)

REPL (read evaluate print loop)

Download the file from

Clone the git repository

git clone https://github.com/manorite/Spark-and-Hadoop-developer-Certification.git

move the files to local file system or HDFS.

Practice 1:

Word count file (ATaleofTwoCities.txt)

Step 1: move to HDFS from the local file system

Step 2: In spark read the file and count the number of words

Step 3: save as text file as solution in practice/solution1.

Solution:

```
val word = sc.textFile("practise/ATaleofTwoCities.txt")
val wordcount = word.flatMap(rec => rec.split(" ")).map(rec => (rec , 1))
wordcount.reduceByKey(_+_).saveAsTextFile("practice/solution1")
```

Problem 2:

Use the classic model

Which orders have a value greater than $5,000 from the classic model database?

Save the result as json file in location practice/solution2

Solution:

Use the path where the data file is store (it can be local file or HDFS location)

```
val orderdetails = sc.textFile("classic_model/datafiles/OrderDetails.txt").
map(rec => (rec.split(",")(0), rec.split(",")(2).toInt * rec.split(",")(3).toFloat)
val ordergreaterthan = orderdetailMap.filter(rec => (rec._2 >5000))
ordergreaterthan.toDF.write.json("practise/solution2")
```

Problem 3:

List top 10 customers from retail_db database? Save the file in parquet format in location practice/solution3

a) **Using Spark RDD and store in location practice/solution3-RDD in Gzip format**
b) **Using Spark SQL Context and store in location practice/solution3-SQL**

Solution:

Part a)

```
val customers = sc.textFile("sqoop_import/customers")

val orders = sc.textFile("sqoop_import/orders")

val order_items = sc.textFile("sqoop_import/order_items")

val customerMap = customers.map(rec => (rec.split(",")(0).toInt, rec))

val ordersMap = orders.map(rec => (rec.split(",")(0).toInt , rec.split(",")(2).toInt))

val order_itemsMap = order_items.map(rec => (rec.split(",")(1).toInt , rec.split(",")(4).toDouble))

val orderJoin = ordersMap.join(order_itemsMap)

val order_format =  orderJoin.map(rec => (rec._2._1 , rec._2._2))

val customerJoin = customerMap.join(order_format)

val customerSort = customerJoin.map(rec => ((rec._2._1.split(",")(1),rec._2._1.split(",")(2)),rec._2._2))

val CustomerDF = customerSort.sortBy(rec => -rec._2).take(10).foreach(println)
```

Part b)

Using SQL Context

```
case class Customers(

customer_id :Int,

customer_first : String,

customer_last : String)

val customerDF = customers.map(rec => {

val r = rec.split(",")

Customers(r(0).toInt, r(1),r(2))

}).toDF
```

```
customerDF.registerTempTable("customers")
val order_items = sc.textFile("sqoop_import/order_items")
case class Order_items (
order_item_order_id : Int,
order_item_subtotal : Double)
val order_itemDF = order_items.map(rec => {
val r = rec.split(",")
Order_items(r(1).toInt , r(4).toDouble)
}).toDF
order_itemDF.registerTempTable("order_items")
val orders = sc.textFile("sqoop_import/orders")
case class Orders(
order_id : Int,
order_date : String,
order_customer_id : Int,
order_status: String)
val orderDF = orders.map(rec => {
Orders(r(0).toInt,r(1),r(2).toInt,r(3))
}).toDF
val orderDF = orders.map(rec => {
val r = rec.split(",")
Orders(r(0).toInt,r(1),r(2).toInt,r(3))
}).toDF
orderDF.registerTempTable("orders")
val df = sqlContext.sql("select c.customer_first, c.customer_last, oi.order_item_subtotal from customers c join orders o on c.customer_id = o.order_customer_id join order_items oi on oi.order_item_order_id = o.order_id order by oi.order_item_subtotal desc limit 10")
df.write.parquet("practise/solution3-SQL")
```

Problem 4:

Using sqoop, import orders table into hdfs to folders practice/problem4/orders. File should be loaded as Avro File and use snappy compression

Using sqoop, import order_items table into hdfs to folders practice/problem4/order_items. Files should be loaded as avro file and use snappy compression

Using Spark Scala load data at practice/problem4/orders and practice/problem4/orders_items items as dataframes.

Expected Intermediate Result: Order_Date , Order_status, total_orders, total_amount.

That is., please find total orders and total amount per status per day. The result should be sorted by order date in descending, order status in ascending and total amount in descending and total orders in ascending. Aggregation should be done using below methods. However, sorting can be done using a dataframe or RDD. Perform aggregation in each of the following ways

a) Use Data Frames API, Spark SQL and/or RDD - here order_date should be YYYY-MM-DD format

Store the result as parquet file into hdfs using gzip compression under folder

b) practice/problem4/result4-gzip

Store the result as parquet file into hdfs using snappy compression under folder

c) practice/problem4/result4-snappy

Store the result as CSV file into hdfs using No compression under folder

d) practice/problem4/result4-csv
e) create a mysql table named result and load data from practice/problem4/result4a-csv to mysql table named result

Solution:

Open Terminal

sqoop import

--connect jdbc:mysql://quickstart.cloudera:3306/retail_db

--username retail_dba

--password cloudera

--table orders

--target-dir practise/problem4/orders

--compress

--compression-codec=Snappy

--as-avrodatafile

sqoop import

--connect jdbc:mysql://quickstart.cloudera:3306/retail_db

--username retail_dba

--password cloudera

--table order_items

--target-dir practise/problem4/order_items

--compress

--compression-codec=Snappy

--as-avrodatafile

import com.databricks.spark.avro._;

val ordersDF = sqlContext.read.avro("practise/problem4/orders")

val order_itemsDF = sqlContext.read.avro("practise/problem4/order_items")

Part a) Using Data frames, Spark SQL and/or RDD

val orderjoinDF = orderDF.join(order_itemsDF,orderDF("order_id") ===
order_itemsDF("order_item_order_id"))

val orderjoinDF_col = orderjoinDF.select("order_date","order_status","order_item_subtotal")

orderjoinDF_col.registerTempTable("orderTable")

val result = sqlContext.sql("select substr(order_date,1,10) Order_Date, order_status ,
count(order_status) total_orders, sum(order_item_subtotal) total_amount from orderTable group by
Order_Date, order_status order by Order_Date desc, order_status , total_amount desc, total_orders")

Part b)

sqlContext.setConf("spark.sql.parquet.compression.codec","gzip");

result.write.parquet("practise/problem4/result4-gzip")

Part c)

sqlContext.setConf("spark.sql.parquet.compression.codec","snappy");

result.write.parquet("practise/problem4/result4-snappy")

part d)

result.map(r => r(0)+ "," + r(1) + "," + r(2) + "," + r(3)).saveAsTextFile("practise/problem4/result4-csv")

part e)

mysql

create table result(Order_Date date, order_status varchar(20), total_orders int, total_amount double);

terminal

sqoop export --table result --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username retail_dba --password cloudera --export-dir practise/problem4/result4-csv

Problem 5:

Using sqoop copy data available in mysql products table to folder practise/products on hdfs as text file. columns should be delimited by pipe '|'

move all the files from practise/products folder to practise/problem5/products folder

Change permissions of all the files under practise/problem5/products such that owner has read,write and execute permissions, group has read and write permissions whereas others have just read and execute permissions

read data in practise/problem5/products and do the following operations using

a) Dataframes API

b) spark SQL

c) RDDs method.

Your solution should have three sets of steps. Sort the resultant dataset by category id filter such that your RDD\DF has products whose price is lesser than 100 USD on the filtered data set find out the highest value in the product_price column under each category on the filtered data set also find out total products under each category on the filtered data set also find out the average price of the product under each category on the filtered data set also find out the minimum price of the product under each category

store the result in parquet file using snappy compression under these folders respectively

practice/problem5/products/result-df

practice/problem5/products/result-sql

practice/problem5/products/result-rdd

Solution

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --table products --target-dir practise/products --fields-terminated-by "|"

hadoop fs -mkdir practise/problem5

hadoop fs -mv practise/products practise/problem5

//Read is 4, Write is 2 and execute is 1.

//ReadWrite,Execute = 4 + 2 + 1 = 7

//Read,Write = 4+2 = 6

//Read ,Execute=4+1=5

hadoop fs -chmod 765 practise/products practise/problem5

hadoop fs -ls practise/problem5/products/*

Part a)

Spark-shell

val products = sc.textFile(“practise/problem5/products”)

case class Products(

product_id : Int,

product_category_id: Int ,

product_name : String ,

product_description : String,

product_price : Double,

product_image : String)

val productDF = products.map(rec => {

val r = rec.split('|')

Products(r(0).toInt,r(1).toInt,r(2),r(3),r(4).toDouble,r(5))

}).toDF

val product_data = productDF.filter("product_price < 100").groupBy("product_category_id").agg(max("product_price").alias("Max_price"),countDistinct("product_price").alias("Total_products"),avg("product_price").alias("Average_price"),min("product_price").alias("Min_price")).sort(desc("product_category_id"))

sqlContext.setConf("spark.sql.parquet.compression.codec","snappy")

product_data.write.parquet("practise/problem5/products/result-df")

Part b)

productDF.registerTempTable("product")

val productSQL = sqlContext.sql("select product_category_id, max(product_price) Max_price ,count(product_price) total_products,avg(product_price) average_price , min(product_price) minimum_price from product where product_price < 100 group by product_category_id order by product_category_id desc")

```
sqlContext.setConf("spark.sql.avro.compression.codec","gzip")

productSQL.write.avro("practise/problem5/products/result-sql")
```

Part c)

```
val productsRDD = products.map(rec => {

val r = rec.split('|')

(r(1).toInt , r(4).toDouble)

}).filter(rec => rec._2 <100)

val productGroup = productsRDD.groupByKey()

val product_table = productGroup.map(rec => (rec._1 , (rec._2.max ,rec._2.size, rec._2.sum/rec._2.size ,rec._2.min)))

product_table.sortBy(rec => -rec._1).map(r => (r._1,r._2._1,r._2._2,r._2._3,r._2._4)).toDF.write.parquet("practise/problem5/products/result-rdd")
```

Problem 6:

- Import orders table from mysql as text file to the destination practise/problem6/text. Fields should be terminated by a tab character ("\t") character and lines should be terminated by new line character ("\n").
- Import orders table from mysql into hdfs to the destination practise/problem6/avro. File should be stored as avro file.
- Import orders table from mysql into hdfs to folders practise/problem6/parquet. File should be stored as parquet file.
- Transform/Convert data-files at practice/problem6/avro and store the converted file at the following locations and file formats
    - save the data to hdfs using snappy compression as parquet file at practise/problem6/parquet-snappy-compress
    - save the data to hdfs using gzip compression as text file at practise/problem6/text-gzip-compress
    - save the data to hdfs using no compression as sequence file at practise/problem6/sequence
    - save the data to hdfs using snappy compression as text file at practice/problem6/text-snappy-compress
- Transform/Convert data-files at practise/problem6/parquet-snappy-compress and store the converted file at the following locations and file formats
    - save the data to hdfs using no compression as parquet file at practise/problem6/parquet-no-compress
    - save the data to hdfs using snappy compression as avro file at practise/problem6/avro-snappy
- Transform/Convert data-files at practise/problem6/avro-snappy and store the converted file at the following locations and file formats
    - save the data to hdfs using no compression as json file at practise/problem6/json-no-compress
    - save the data to hdfs using gzip compression as json file at practise/problem6/json-gzip
- Transform/Convert data-files at practise/problem6/json-gzip and store the converted file at the following locations and file formats
    - save the data to as comma separated text using gzip compression at practise/problem6/csv-gzip
- Using spark access data at practise/problem6/text and stored it back to hdfs using no compression as ORC file to HDFS to destination practise/problem6/orc
- Using spark access data at practise/problem6/orc and compress using snappy codec and store as practise/problem6/orc-snappy

Solution:

```
sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --table orders --fields-terminated-by "\t" --lines-terminated-by "\n" --target-dir "practise/problem6/text"

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --table orders --target-dir "practise/problem6/avro" --as-avrodatafile

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --table orders --target-dir "practise/problem6/parquet" --as-parquetfile

import com.databricks.spark.avro._

val order_avro = sqlContext.read.avro("practise/problem6/avro")

sqlContext.setConf("spark.sql.parquet.compression.codec","snappy")

order_avro.write.parquet("practise/problem6/parquet-snappy-compress")

order_avro.map(x=> x(0)+"\t"+x(1)+"\t"+x(2)+"\t"+x(3)).saveAsTextFile("practise/problem6/text-gzip-compress",classOf[org.apache.hadoop.io.compress.GzipCodec])

order_avro.map(x=> (x(0).toString,x(0)+"\t"+x(1)+"\t"+x(2)+"\t"+x(3))).saveAsSequenceFile("practise/problem6/Sequence")

order_avro.map(x => x(0) + "\t" + x(1) + "\t" + x(2) + "\t" + x(3)).saveAsTextFile("practise/problem6/text-snappy-compress",classOf[org.apache.hadoop.io.compress.SnappyCodec])

val order_snappy = sqlContext.read.parquet("practise/problem6/parquet-snappy-compress")

order_snappy.write.parquet("practise/problem6/parquet-no-compress")

sqlContext.setConf("spark.sql.avro.compression.codec","snappy")

order_snappy.write.avro("practise/problem6/avro-snappy")

val order_avro = sqlContext.read.avro("practise/problem6/avro-snappy")

order_avro.write.json("practise/problem6/json-no-compress")
```

```
#sqlContext.setConf("spark.sql.json.compression.codec","gzip")

order_avro.write.json("practise/problem6/json-gzip")


val orderMap = order_json_gzip.map(x => {

(x(0) + "," + x(1) + "," + x(2) + "," + x(3))

})

orderMap.saveAsTextFile("practise/problem6/csv-
gzip",classOf[org.apache.hadoop.io.compress.GzipCodec])


val order_text = sc.textFile("practise/problem6/text").toDF

order_text.write.orc("practise/problem6/orc")


sqlContext.setConf("spark.sql.orc.compression.codec","SnappyCodec")

order_orc.write.orc("practise/problem6/orc-snappy")
```

Problem 7

1) Create a hive meta store database named problem7 and import all tables from mysql retail_db database into hive meta store.
2) On spark shell use data available on meta store as source and perform next steps. [this proves your ability to use meta store as a source]
3) Rank products within department by price and order by department ascending and rank descending [this proves you can produce ranked and sorted data on joined data sets]
4) find top 10 customers with most unique product purchases. if more than one customer has the same number of product purchases then the customer with the lowest customer_id will take precedence [this proves you can produce aggregate statistics on joined datasets]
5) On dataset from step 3, apply filter such that only products less than 100 are extracted [this proves you can use subqueries and filter data]
6) On dataset from step 4, extract details of products purchased by top 10 customers which are priced at less than 100 USD per unit [this proves you can use subqueries and also filter data]
7) Store the result of 5 and 6 in new meta store tables within hive. [this proves your ability to use metastore as a sink]

Hive

Create database problem7;

Terminal

sqoop import-all-tables --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --hive-import --hive-database problem7 --warehouse-dir practise/problem7

create a soft link

sudo ln -s /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/hive-site.xml

hive

set hive.auto.convert.join.noconditionaltask =false

var hc = new org.apache.spark.sql.hive.HiveContext(sc);

hc.sql("show databases").show

hc.sql("use problem7")

val Result1 = hc.sql("select p.product_id, p.product_name, d.department_name,p.product_price, rank() over (partition by p.product_price order by d.department_name) rnk, dense_rank() over (partition by p.product_price order by d.department_name desc) drnk from products p join categories c on c.category_id = p.product_category_id join departments d on c.category_department_id = d.department_id inner join order_items oi on oi.order_item_product_id = p.product_id").show(200)

```
val hiveResult = hc.sql("select c.customer_id, concat(c.customer_fname, ' ' , c.customer_lname) as full_name, cast(sum(oi.order_item_subtotal) as decimal(10,2)) as total, count(distinct(oi.order_item_product_id)) as U_count from customers c join orders o on c.customer_id = o.order_customer_id join order_items oi on o.order_id = oi.order_item_order_id group by c.customer_id, concat(c.customer_fname,' ' , c.customer_lname) order by u_count desc limit 10")


Result1.registerTempTable("table1")

val result1 = hc.sql("select * from table1 where product_price < 100").show

hiveResult.registerTempTable("table2")

val result2 = hc.sql("select p.* from products p join order_items oi on oi.order_item_product_id = p.product_id join orders o on o.order_id = oi.order_item_order_id join table2 tb on tb.customer_id = o.order_customer_id where p.product_price < 100")

result1.registerTempTable("result1")

hc.sql("create table problem7.result1 as select * from result1")

result2.registerTempTable(“result2”)

hc.sql("create table problem7.result2 as select * from result2")
```

Problem 8:

1. This step comprises of three substeps. Please perform tasks under each subset completely
    a. using sqoop pull data from MYSQL orders table into practise/problem8/prework as AVRO data file using only one mapper
    b. Pull the file from practise\problem7\prework into a local folder named flume-avro
    c. create a flume agent configuration such that it has an avro source at localhost and port number 11113, a jdbc channel and an hdfs file sink at practise/problem8/sink
    d. Use the following command to run an avro client flume-ng avro-client -H localhost -p 11113 -F <<Provide your avro file path here>>
2. The CDH comes prepackaged with a log generating job. start_logs, stop_logs and tail_logs. Using these as an aid and provide a solution to below problem. The generated logs can be found at path /opt/gen_logs/logs/access.log
    a. run start_logs
    b. write a flume configuration such that the logs generated by start_logs are dumped into HDFS at location practise/problem7/step2. The channel should be non-durable and hence fastest in nature. The channel should be able to hold a maximum of 1000 messages and should commit after every 200 messages.
    c. Run the agent.
    d. confirm if logs are getting dumped to HDFS
    e. run stop_logs.

Solution:

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/retail_db --username retail_dba --password cloudera --table orders --target-dir practise/problem8/prework --as-avrodatafile -m 1

hadoop fs -get practise/problem8/prework flume-avro

vi flumeConf.conf

conf File

Name the components on this agent

res1.sources = r1

res1.sinks = k1

res1.channels = c1

Describe/configure the source

res1.sources.r1.type = avro

res1.sources.r1.bind = localhost

res1.sources.r1.port = 11113

Describe the sink

res1.sinks.k1.type = hdfs

res1.sinks.k1.hdfs.path = practise/problem8/sink

Use a channel which buffers events in memory

res1.channels.c1.type = memory

res1.channels.c1.capacity = 1000

res1.channels.c1.transactionCapacity = 100

Bind the source and sink to the channel

res1.sources.r1.channels = c1

res1.sinks.k1.channel = c1

flume-ng agent --name res1 --conf /home/cloudera --conf-file flumeConf.conf

flume-ng avro-client -H localhost -p 11113 -F /home/cloudera/flume-avro/part-m-00000.avro

Validate

hadoop fs -ls practise/problem8/sink

Step 2

vi flumeConf2.conf

#Components on this agent

res2.sources = r1

res2.sinks = k1

res2.channels = c1

Describe/configure the source

```
res2.sources.r1.type = exec
res2.sources.r1.bind = localhost
res2.sources.r1.command = tail -F /opt/gen_logs/logs/access.log

# Describe the sink
res2.sinks.k1.type = hdfs
res2.sinks.k1.hdfs.path = practise/problem8/step2

# Use a channel which buffers events in memory
res2.channels.c1.type = memory
res2.channels.c1.capacity = 1000
res2.channels.c1.transactionCapacity = 200

# Bind the source and sink to the channel
res2.sources.r1.channels = c1
res2.sinks.k1.channel = c1
```

```
start_logs
flume-ng agent --name res2 --conf /home/cloudera --conf-file flumeConf2.conf
#validate
hadoop fs -ls practise/problem8/step2
stop_logs
```

Problem 9:

- create a hive meta store database named problem9 and import table from mysql classic database into hive meta store.
- On spark shell use data, available on meta store as source and perform next steps.
- Get lowest 3 customer within country by amount and order by price ascending and dense_rank descending [this proves you can produce ranked and sorted data on joined data sets]
- On dataset store the result in new meta store tables within hive. [this proves your ability to use metastore as a sink]

spark-shell

hc.sql("create database problem9")

terminal

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table customers --hive-import --hive-database problem9 --create-hive-table

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table payments --hive-import --hive-database problem9 --create-hive-table

spark-shell

hc.sql("use problem9")

val hiveresult = hc.sql("select customerName, country , total, rank() over (partition by country order by total) rnk, dense_rank() over (partition by country order by total desc) drnk from (select c.customerName, c.country, cast(sum(p.amount) as decimal(10,2)) total from customers c join payments p on c.customerNumber = p.customerNumber group by c.customerName, c.country order by total desc) a order by country desc")

val hiveResults = hiveresult.filter("rnk < 4")

hiveResults.registerTempTable("hiveResults")

hc.sql("create table results as select * from hiveResults")

Problem 10:

Use the classic database

Compute the commission for each sales representative, assuming the commission is 5% of the value of an order. Sort by employee last name and first name.

Store the data in practise/problem10/result-avro in avro format and snappy compression

Store the data in practise/problem10/text in GzipCodec format

Who reports to Mary Patterson?

Solution

Spark-shell

hc.sql("create database problem10")

terminal

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table orderdetails --hive-import --hive-database problem10 --create-hive-table

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table orders --hive-import --hive-database problem10 --create-hive-table

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table customers --hive-import --hive-database problem10 --create-hive-table

sqoop import --connect jdbc:mysql://quickstart.cloudera:3306/classic --username classic_dba --password classic --table employees --hive-import --hive-database problem10 --create-hive-table

spark-shell

val hc = new org.apache.spark.sql.hive.HiveContext(sc)

hc.sql("use problem10")

val result = hc.sql("select e.employeeNumber,e.lastName,e.firstName,e.officeCode, cast(sum(0.05 * od.priceEach * od.quantityOrdered) as decimal(10,2)) commision from employees e join customers c on c.salesRepEmployeeNumber = e.employeeNumber join orders o on o.customerNumber = c.customerNumber join orderdetails od on od.orderNumber = o.orderNumber group by e.employeeNumber,e.lastName,e.firstName,e.officeCode order by e.lastName,e.firstName")

val Result = result.map(x => x(0) + "," + x(1) + "," + x(2) + "," + x(3) + "," + x(4))

Result.saveAsTextFile("practise/problem10/text",classOf[org.apache.hadoop.io.compress.GzipCodec])

hc.sql("use problem10")

```
val result1 = hc.sql("select concat(firstName ," ",lastName) full_name from employees e where e.reportsTo = 1056”)

result1.registerTempTable("result1")

hc.sql("create table Res1 as select * from result1")
```