**CCA Exam-assignments**

Saturday, 7 November 2020

3:45 PM

**spark-shell --packages org.apache.spark:spark-avro\_2.12:2.4.5**

hdfs dfs -ls /user/spark/dataset/retail\_db | sed '1d;s/ \*/ /g'| cut -d\ -f8;

spark.sqlContext.setConf("hive.exec.dynamic.partition", "true")

spark.sqlContext.setConf("hive.exec.dynamic.partition.mode", "nonstrict")

|  |  |  |
| --- | --- | --- |
| # | Input | Output |
| 1 |  | **Typed schema**  customer\_id int,customer\_fname string,customer\_lname string,customer\_email string,customer\_passcode string,customer\_street string,customer\_city string,customer\_state string,customer\_zipcode string    **Read data**  scala> val dataDF1 = spark.read.option("delimiter", "\t").csv("/user/spark/dataset/retail\_db/customers-tab-delimited").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")    **Write data:**    **Solution 1**    scala> dataDF1.filter($"customer\_state" ==="CA").withColumn("customer\_name", concat\_ws(" ", $"customer\_fname", $"customer\_lname")).select("customer\_name").write.text("/user/spark/dataset/result/scenario1/solution")    **Solution 2**  scala> import org.apache.spark.sql.\_  import org.apache.spark.sql.types.\_    scala> val result1 = spark.sql("""select concat\_ws(' ', customer\_fname, customer\_lname) customer\_name from dataView1 where customer\_state='CA'""")    scala> result1.write.mode(SaveMode.Overwrite).text("/user/spark/dataset/result/scenario1/solution")    hdfs dfs -tail /user/spark/dataset/result/scenario1/solution/part-00000-94942cbc-b1b5-41a0-b5ac-6d3165f0b8fa-c000.txt |
| 2 |  | val dataParquet1 = spark.read.parquet("/user/spark/dataset/retail\_db/orders\_parquet")    **Solution 1**  scala> dataParquet1.filter($"order\_status"==="COMPLETE").withColumn("order\_date", to\_date(from\_unixtime($"order\_date"/1000))).select("order\_id", "order\_date", "order\_status").write.mode(SaveMode.Overwrite).option("compression","gzip").json("/user/spark/dataset/result/scanario2/solution")    **Solution 2**  scala> dataParquet1.createOrReplaceTempView("dataParquet1View")    spark.sql("""select order\_id, to\_date(from\_unixtime(order\_date/1000)) order\_date, order\_status from dataParquet1View where order\_status = 'COMPLETE'""").show(5)    hdfs dfs -ls /user/spark/dataset/result/scanario2/solution |
| 3 |  | **Read data**  scala> val dataDF3 = spark.read.option("inferschema", true).option("delimiter", "\t").csv("/user/spark/dataset/retail\_db/customers-tab-delimited").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")    **Solution**    val dataResult3 = dataDF3.filter($"customer\_city" === "Caguas")    scala> dataResult3.write.option("compression", "snappy").orc("/user/spark/dataset/result/scenario3/solution")      --view orc file and compression details    hencilpeter@cca-175-cluster-m:~/retail\_db\_dataset$ hive --orcfiledump /user/spark/dataset/result/scenario3/solution/part-00000-3f273c0d-004e-499b-8c69-e64e9a377565-c000.snappy.orc |
| 4 |  | **Read data**  scala> val dataDF4 = spark.read.option("inferschema", true).option("delimiter",",").csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id", "category\_deparment\_id", "category\_name")      Solution    scala> val resultDF4 = dataDF4.map( row => row.mkString( "\t"))      scala> resultDF4.write.option("compression", "lz4").text("/user/spark/dataset/result/scenario4/solution")    --verify the result  hdfs dfs -ls /user/spark/dataset/result/scenario4/solution |
| 5 |  | **spark-shell --packages org.apache.spark:spark-avro\_2.12:2.4.5**    --read data  scala> val dataDF5 = spark.read.format("avro").load("/user/spark/dataset/retail\_db/products\_avro")    --write data  scala> dataDF5.filter($"product\_price" > 1000.0 ).write.option("compression", "snappy").parquet("/user/spark/dataset/result/scenario5/solution")    --verify  hdfs dfs -ls /user/spark/dataset/result/scenario5/solution |
| 6 |  | scala> import org.apache.spark.sql.\_  import org.apache.spark.sql.types.\_    --read. --1  scala> val dataDF61 = spark.sql("select \* from default.orders where substring(order\_date,0,7) >= '2013-01' and substring(order\_date,0,7) <= '2013-12' ")  dataDF61: org.apache.spark.sql.DataFrame = [order\_id: int, order\_date: string ... 2 more fields]    scala> dataDF61.count()  res31: Long = 30662    scala> val dataDF62 = spark.sql("select \* from default.orders where order\_date between '2013-01-01 00:00:00.0' and '2013-12-31 00:00:00.0' ")  dataDF62: org.apache.spark.sql.DataFrame = [order\_id: int, order\_date: string ... 2 more fields]    scala> dataDF62.count()  res32: Long = 30662  --write  cala> dataDF61.write.option("compression", "gzip").parquet("/user/spark/dataset/result/scenario6/solution")    --verify  hdfs dfs -ls /user/spark/dataset/result/scenario6/solution    hencilpeter@cca-175-cluster-m:~/retail\_db\_dataset/util$ hadoop jar parquet-tools-1.9.0.jar cat --json /user/spark/dataset/result/scenario6/solution/part-00008-040b508c-34d2-4eeb-a3ab-12e22d7e1653-c000.gz.parquet |
| 7 |  | --read  val dataDF7 = spark.read.option("delimiter",",").csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id", "category\_deparment\_id", "category\_name")    --write  dataDF7.write.format("hive").option("compression","uncompressed").mode("overwrite").saveAsTable("default.categories\_replica")    --verify  hive> show tables;  hive> select \* from categories\_replica limit 10;  hive> describe formatted categories\_replica; |
| 8 |  | --read  val dataDF8 = spark.read.option("delimiter",",").csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id", "category\_deparment\_id", "category\_name")    --write  scala> spark.sql("create table if not exists default.categories\_parquet(category\_id int, category\_name string) stored as parquet")    scala> dataDF8.select("category\_id", "category\_name").write.mode("overwrite").saveAsTable("categoris\_parquet")    --verify  hive> describe formatted categories\_parquet; |
| 9 |  | --read  val dataDF = spark.read.format("avro").option("compression","snappy").load("/user/spark/dataset/retail\_db/products\_avro")    --write  scala> dataDF.select("product\_id","product\_price").write.option("compression","none").json("user/spark/dataset/result/scenario9/solution")    --verify  hdfs dfs -ls user/spark/dataset/result/scenario9/solution |
| 10 |  | scala> spark.sqlContext.setConf("hive.exec.dynamic.partition", "true")  scala> spark.sqlContext.setConf("hive.exec.dynamic.partition.mode", "nonstrict")    --read  scala> val dataDF10 = spark.read.option("delimiter",",").csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id", "category\_deparment\_id", "category\_name")    --write  scala> dataDF10.write.format("hive").partitionBy("category\_deparment\_id").saveAsTable("default.categories\_partitioned")    --verify  hive> select \* from categories\_partitioned limit 10; |
| 11 |  | --read  scala> val dataDF11 = spark.read.format("avro").load("/user/spark/dataset/retail\_db/customers-avro")    **--write**  val resultDF11 = dataDF11.withColumn("customer\_name", concat\_ws(" ", substring($"customer\_fname", 0, 1), $"customer\_lname"))    scala> resultDF11.select("customer\_id", "customer\_name").map( row => row.mkString("\t")).write.option("compression","bzip2").text("/user/spark/dataset/result/scenario11/solution") |
| 12 |  | --read  val dataDF12 = spark.read.parquet("/user/spark/dataset/retail\_db/orders\_parquet")  **--write**  val resultDF12 = dataDF12.withColumn("order\_date", to\_date( from\_unixtime($"order\_date"/1000))).filter($"order\_status" === "PENDING" && substring($"order\_date",0,7)==="2013-07")    scala> resultDF12.select("order\_date","order\_status").write.option("compression","snappy").json("/user/spark/dataset/result/scenario12/solution")    **--another filtering approach**  **dataDF12.filter(col("order\_date").like("%2013-07%")).select("order\_date","order\_status")**    **--verify**  dfs dfs -ls /user/spark/dataset/result/scenario12/solution |
| 13 |  | --read  scala> val dataDF13 = spark.sql("select \* from default.customers")  **--write**  scala> dataDF13.createOrReplaceTempView("dataDF13View")    val resultDF13 = spark.sql("select \* from dataDF13View where customer\_fname like '%Rich%' ")    resultDF13.write.option("compression", "snappy").parquet("/user/spark/dataset/result/scenario13/solution")    **--verify**  hdfs dfs -ls /user/spark/dataset/result/scenario12/solution |
| 14 |  | **--read**  scala> val dataDF14 = spark.read.option("inferschema", true).option("delimiter", "\t").csv("/user/spark/dataset/retail\_db/customers-tab-delimited").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")      **--write**  dataDF14.createOrReplaceTempView("dataDF14View")    scala> val resultDF14 = spark.sql("select customer\_state, count(1) count from dataDF14View where customer\_fname like 'M%' group by customer\_state")    scala> resultDF14.write.option("compression", "gzip").parquet("/user/spark/dataset/result/scenario14/solution")      **--verify**  hdfs dfs -ls /user/spark/dataset/result/scenario14/solution    hencilpeter@cca-175-cluster-m:~/retail\_db\_dataset/util$ hadoop jar parquet-tools-1.9.0.jar cat --json /user/spark/dataset/result/scenario14/solution/part-00199-9373f773-1e3c-41d8-966d-5dc962372b37-c000.gz.parque |
| 15 |  | --read  val dataOrdersDF15 = spark.read.csv("/user/spark/dataset/retail\_db/orders").toDF("order\_id", "order\_date","order\_customer\_id", "order\_status")    val dataCustomersDF15 = spark.read.csv("/user/spark/dataset/retail\_db/customers").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")  **---write**  scala> dataOrdersDF15.createOrReplaceTempView("dataOrdersDF15View")    scala> dataCustomersDF15.createOrReplaceTempView("dataCustomersDF15View")        scala> spark.sql("select o.order\_customer\_id, c.customer\_fname, c.customer\_lname, count(1) count from dataCustomersDF15View c join dataOrdersDF15View o on c.customer\_id = o.order\_customer\_id where c.customer\_fname like 'M%' group by o.order\_customer\_id, c.customer\_fname, c.customer\_lname having count(1) > 5 order by count desc").select("customer\_fname","customer\_lname", "count").map(row => row.mkString("|")).write.option("compression","gzip").text("/user/spark/dataset/result/scenario15/solution") |
| 16 |  | --read  scala> val dataDF16 = spark.read.csv("/user/spark/dataset/retail\_db/orders").toDF("order\_id", "order\_date","order\_customer\_id", "order\_status")      **--write**  dataDF16.withColumn("order\_date",substring($"order\_date",0,10)).filter($"order\_status" === "SUSPECTED\_FRAUD").show(5)    dataDF16.createOrReplaceTempView("dataDF16View")    scala> spark.sql("select substring(order\_date,0,7) order\_date, count(1) count from dataDF16View where order\_status = 'SUSPECTED\_FRAUD' group by substring(order\_date,0,7) order by order\_date desc").show(5) |
| 17 |  | --read  scala> val dataDFProducts17 = spark.read.csv("/user/spark/dataset/retail\_db/products").toDF("product\_id","product\_category\_id","product\_name","product\_description","product\_price","product\_image")    scala> val dataDFCategories17 = spark.read.csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id","category\_department\_id","category\_name")    scala> dataDFProducts17.createOrReplaceTempView("dataDFProducts17View")  scala> dataDFCategories17.createOrReplaceTempView("dataDFCategories17View")    **--write**    scala> spark.sql("select c.category\_name, round(max(p.product\_price),2) max\_price, round(min(p.product\_price),2) min\_price, round(avg(p.product\_price),2) avg\_price from dataDFProducts17View p join dataDFCategories17View c on p.product\_category\_id = c.category\_id group by p.product\_category\_id, c.category\_name").write.option("compression","deflate").json("/user/spark/dataset/result/scenario17/solution")    **--verify**  hdfs dfs -ls /user/spark/dataset/result/scenario17/solution |
| 18 |  | --read  scala> val dataOrdersDF18 = spark.read.csv("/user/spark/dataset/retail\_db/orders").toDF("order\_id", "order\_date","order\_customer\_id", "order\_status")    scala> val dataCustomersDF18 = spark.read.csv("/user/spark/dataset/retail\_db/customers").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")    **--write**  dataOrdersDF18.createOrReplaceTempView("dataOrdersDF18View") dataCustomersDF18.createOrReplaceTempView("dataCustomersDF18View")    **Spark.sql("select c.customer\_id, c.customer\_fname, c.customer\_lname, count(1) count from** dataCustomersDF18 c join dataOrdersDF18 o on c.customer\_id = o.order\_customer\_id where substring(o.order\_date,0,4) = '2014' group by c.customer\_id, c.customer\_fname, c.customer\_lname order by count desc").show(5)15/11/20 |
| 19 |  | --read  scala> val dataDFProducts19 = spark.read.csv("/user/spark/dataset/retail\_db/products").toDF("product\_id","product\_category\_id","product\_name","product\_description","product\_price","product\_image")    scala> val dataDFCategories19 = spark.read.csv("/user/spark/dataset/retail\_db/categories").toDF("category\_id","category\_department\_id","category\_name")    val dataDFOrderItems19 = spark.read.csv("/user/spark/dataset/retail\_db/order\_items").toDF("order\_item\_id", "order\_item\_order\_id","order\_item\_product\_id", "order\_item\_quantity", "order\_item\_subtotal", "order\_item\_product\_price"))    **--write**  dataDFProducts19 .createOrReplaceTempView("dataDFProducts19View")  dataDFCategories19.createOrReplaceTempView("dataDFCategories19View")  dataDFOrderItems19.createOrReplaceTempView("dataDFOrderItems19View")    scala> val resultDF19 = spark.sql("select c.category\_name, p.product\_name, round(sum(o.order\_item\_subtotal), 2) product\_revenue, rank() over (partition by c.category\_name, p.product\_name order by round(sum(o.order\_item\_subtotal), 2) desc ) rank from dataDFProducts19View p join dataDFCategories19View c on p.product\_category\_id =c.category\_id join dataDFOrderItems19View o on p.product\_id = o.order\_item\_product\_id where c.category\_name= 'Accessories' group by c.category\_name, p.product\_name order by product\_revenue desc" )    resultDF19.select("category\_name", "product\_name", "product\_revenue").map( row => row.mkString("|")).write.text("/user/spark/dataset/result/scenario19/solution") |
| 20 |  | --read  val dataCustomersDF20 = spark.read.csv("/user/spark/dataset/retail\_db/customers").toDF("customer\_id","customer\_fname","customer\_lname","customer\_email","customer\_passcode","customer\_street","customer\_city","customer\_state","customer\_zipcode")    val dataOrdersDF20 = spark.read.csv("/user/spark/dataset/retail\_db/orders").toDF("order\_id", "order\_date","order\_customer\_id", "order\_status")    val dataDFOrderItems20 = spark.read.csv("/user/spark/dataset/retail\_db/order\_items").toDF("order\_item\_id", "order\_item\_order\_id","order\_item\_product\_id", "order\_item\_quantity", "order\_item\_subtotal", "order\_item\_product\_price")    --write  dataCustomersDF20.createOrReplaceTempView("dataCustomersDF20View")  dataOrdersDF20.createOrReplaceTempView("dataOrdersDF20View")  dataDFOrderItems20.createOrReplaceTempView("dataDFOrderItems20View")    val resultDF20 = spark.sql("select c.customer\_id, c.customer\_city, c.customer\_fname, c.customer\_lname, round(sum(oi.order\_item\_subtotal), 2) product\_revenue from dataCustomersDF20View c join dataOrdersDF20View o on c.customer\_id = o.order\_customer\_id join dataDFOrderItems20View oi on o.order\_id = oi.order\_item\_id group by c.customer\_id, c.customer\_city, c.customer\_fname, c.customer\_lname having round(sum(oi.order\_item\_subtotal), 2) > 500 ").select("customer\_fname", "customer\_lname", "product\_revenue")    resultDF20.write.option("compression", "snappy").parquet("/user/spark/dataset/result/scenario20/solution")    scala> resultDF20.write.option("compression", "snappy").parquet("/user/spark/dataset/result/scenario20/solution") |
|  |  |  |
|  |  |  |