**About the project:**

This project contains scala with spark processing tool for Refinitiv Matching Engine Exercise with the below problem statement.

==================**Refinitiv Matching Engine Exercise**===================

Your task is to create a new matching engine for FX orders. The engine will take a CSV file of orders for a given

currency pair and match orders together. In this example you'll be looking at USD/GBP.

There are two types of orders, BUY and SELL orders. A BUY order is for the price in USD you'll pay for GBP, SELL

order is the price in USD you'll sell GBP for.

Each order has the following fields:

1. Order ID

- This is a unique ID in the file which is used to track an order

2. User Name

- This is the user name of the user making the order

3. Order Time

- This is the time, in milliseconds since Jan 1st 1970, the order was placed

4. Order Type

- Either BUY or SELL

5. Quantity

- The number of currency units you want to BUY or SELL

6. Price

- The price you wish to sell for, this is in the lowest unit of the currency, i.e. for GBP it's in pence and for USD it's cents

The matching engine must do the following:

- It should match orders when they have the same quantity

- If an order is matched it should be closed

- If an order is not matched it should be kept on an "order book" and wait for an order which does match

- When matching an order to the book the order should look for the best price

- The best price for a BUY is the lowest possible price to BUY for

- The best price for a SELL is the highest possible price to SELL for

- You should always use the price from the "order book" when matching orders

- When an order has matched you should record the IDs of both orders, the price, quantity and time of the match

- If two orders match with the same price the first order is used

- Orders won't have the same timestamp

The file exampleOrders.csv is some example trading data, the matches for that trading data is in outputExampleMatches.csv

**Build Tool: Maven**

**Scala Version: Org.scala-lang: scala-library:2.11.11**

**Sample Command to run the application locally:**

Spark-submit --master yarn --deploy-mode local --class pack.obj /home/cloudera/ ss-0.0.1-SNAPSHOT.jar

**Sample Command to run the application on yarn with 6 nodes 32 Vcores 64GB RAM:**

Spark-submit --master yarn --deploy-mode cluster --executor-cores 6 --executor-memory 5g --num-executors 40 --class pack.obj /home/cloudera/ ss-0.0.1-SNAPSHOT.jar

=============please find the below code as per the requirement=============

**package** pack

**import** org.apache.spark.SparkContext // rdd

**import** org.apache.spark.sql.SparkSession // dataframe

**import** org.apache.spark.SparkConf

**import** org.apache.spark.sql.\_

**import** org.apache.spark.sql.types.\_

**import** org.apache.spark.sql.expressions.Window

**import** org.apache.spark.sql.functions.\_

**object** CodePratice {

**def** main(args:Array[*String*]):Unit={

**val** conf = **new** SparkConf().setMaster("local[\*]").setAppName("first")

**val** sc = **new** SparkContext(conf)

sc.setLogLevel("ERROR")

**val** spark = SparkSession.builder.config(conf).getOrCreate()

**import** spark.implicits.\_

**val** schema = **StructType**(Array(

**StructField**("Order\_ID",**StringType**,**true**),

**StructField**("User\_Name",**StringType**,**true**),

**StructField**("Order\_Time",**StringType**,**true**),

**StructField**("Order\_Type", **StringType**, **true**),

**StructField**("Quantity", **StringType**, **true**),

**StructField**("Price", **StringType**, **true**)

))

**val** buydf = spark

.read

.format("csv")

.schema(schema)

.load("file:///C:/coding\_excercise/exampleOrders.csv")

buydf.show()

println("======BUY Data=========")

**val** dfbuy=buydf.where(buydf("Order\_Type")==="BUY")

.withColumnRenamed("Order\_ID", "Buy\_Order\_ID")

.withColumnRenamed("User\_Name", "Buy\_User\_Name")

.withColumnRenamed("Order\_Time","Buy\_Order\_Time")

.withColumnRenamed("Order\_Type","Buy\_Order\_Type")

.withColumnRenamed("Quantity","Buy\_Quantity")

.withColumnRenamed("Price", "Buy\_Price")

dfbuy.show()

println("======SELL Data=========")

**val** dfsell=buydf.where(buydf("Order\_Type")==="SELL")

.withColumnRenamed("Order\_ID", "Sell\_Order\_ID")

.withColumnRenamed("User\_Name", "Sell\_User\_Name")

.withColumnRenamed("Order\_Time","Sell\_Order\_Time")

.withColumnRenamed("Order\_Type","Sell\_Order\_Type")

.withColumnRenamed("Quantity","Sell\_Quantity")

.withColumnRenamed("Price", "Sell\_Price")

dfsell.show()

println("======Matched Order Data=========")

**val** dforder = dfbuy.join(dfsell,dfbuy("Buy\_Quantity") === dfsell("Sell\_Quantity"),"inner")

dforder.show()

println("======Un-Matched Order Data=========")

**val** dforderunmatched1 = dfbuy.join(dfsell,dfbuy("Buy\_Quantity") === dfsell("Sell\_Quantity"),"left")

.select("Buy\_Order\_ID", "Buy\_User\_Name","Buy\_Order\_Time","Buy\_Order\_Type","Buy\_Quantity","Buy\_Price","Sell\_Order\_ID")

**val** buyorderum =dforderunmatched1.where(dforderunmatched1("Sell\_Order\_ID") isNull)

.withColumnRenamed("Buy\_Order\_ID","Order\_ID")

.withColumnRenamed("Buy\_User\_Name", "User\_Name")

.withColumnRenamed("Buy\_Order\_Time", "Order\_Time")

.withColumnRenamed("Buy\_Order\_Type" , "Order\_Type")

.withColumnRenamed("Buy\_Quantity", "Quantity")

.withColumnRenamed("Buy\_Price" , "Price")

.drop("Sell\_Order\_ID")

buyorderum.show()

**val** dforderunmatched2 = dfbuy.join(dfsell,dfbuy("Buy\_Quantity") === dfsell("Sell\_Quantity"),"right")

.select("Sell\_Order\_ID", "Sell\_User\_Name","Sell\_Order\_Time","Sell\_Order\_Type","Sell\_Quantity","Sell\_Price","Buy\_Order\_ID")

**val** buyorderum1 =dforderunmatched2.where(dforderunmatched1("Buy\_Order\_ID") isNull)

.withColumnRenamed("Sell\_Order\_ID","Order\_ID")

.withColumnRenamed("Sell\_User\_Name", "User\_Name")

.withColumnRenamed("Sell\_Order\_Time", "Order\_Time")

.withColumnRenamed("Sell\_Order\_Type" , "Order\_Type")

.withColumnRenamed("Sell\_Quantity", "Quantity")

.withColumnRenamed("Sell\_Price" , "Price")

.drop("Buy\_Order\_ID")

buyorderum1.show()

**val** finalumorders = buyorderum1.union(buyorderum)

finalumorders.show()

}

}

=====================Using EMR in AWS================

**To achieve the above requirements, we started with necessary details:**

1. Read the given data
2. Create Buy & Sell data frames separately.
3. Create the dataset with order matching and un-matching.

**First, we started with development:**

1. Go to EMR from respective project & Create Cluster with Launch Mode as “Cluster” (3 Nodes Cluster).
2. Provide the necessary details & launch the Cluster.
3. Go to the DNS master URL and open the putty & log in with keypair.ppk.
4. Open spark-shell with required jars .

Ex: spark-shell --packages if any extra jars required

* Start the development in spark-shell step by step & test the code and make sure that our code is working fine.
* And create the project in eclipse by adding necessary jars and create the objects for each step respectively.
* Terminate the Cluster once the development is done.

**Second, develop the Deployment Script:**

1. Create the jar file using Eclipse/Intelliji.
2. Copy the jar file to required location.
3. Create the EMR Cluster with Launch Mode as “Step Execution” by configuring required spark-submit steps separately.
4. Once the Create Cluster is done, Run the spark submit steps and terminate automatically.

**Third, Automated:**

1. Go to the deployment script from "AWS CLI export" which is final shell script and run it on EMR MASTER NODE.
2. Once the EMR Cluster is started, Run the spark-submit steps and terminate it automatically.
3. Verify the data in target location.

**Fourth, Production Deployment:**

1. Commit the code in GIT and give the permission to master branch.
2. DevOps team will build the jar from GIT UI and maintain the Jenkins files & CI/CD pipeline will be built uisng jenkins and make sure that jar file is available on PROD.
3. Provide the executable script to DevOps team for scheduling it from PROD EC2 and DevOps team will copy the executable script from dev s3 location to PROD EC2 or they will download it to local and copy to PROD EC2 via Winscp.
4. Finally the DevOps team will schedule the script using crontab -e.