# Trade Execution Design Document

## Introduction:

The TradeExecution code is responsible for processing trade orders and matching them between an order book and an order file. It reads data from CSV files, performs matching operations, and produces output files. The code is implemented using Spark and Scala.

## Overview:

Need to build a 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’s 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

## Design Consideration:

### 1. Goals :

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

2.Assumptions:

1.The Processing Engine will run in batch processing mode.

2.When There is a match in both order Book and order File then the priority should be given to order Book

3. Best price matching is applicable for only orders from Order Book

3.Dependencies:

The code has dependencies on the following libraries:

• java.util.Calendar and java.text.SimpleDateFormat: Used for generating timestamps for file archival.

• org.apache.spark.sql.\_: Used for Spark SQL operations.

• org.apache.spark.sql.expressions.\_: Used for window functions and expressions in Spark SQL.

• org.apache.spark.sql.types.\_: Used for defining schema types in Spark SQL.

• org.apache.spark.sql.functions.\_: Used for built-in functions in Spark SQL.

• org.apache.hadoop.fs.{FileSystem, Path, FileUtil}: Used for interacting with the Hadoop file system.

### 4.Psuedo Code:

Step-1 : Import the required libraries and packages.

Step-2 : Define an object named "TradeExecution" for the main program.

Step-3 : Create a Spark session and configure it.

Step-4 : Define a function named "readCsvFile" that takes a file path and schema as input and reads the CSV file into a DataFrame using the provided schema.

Step-5 : Define a function named "archiveFile" that moves a file from the source path to the destination path.

Step-6 : Set up variables for the current timestamp using SimpleDateFormat and Calendar.

Step-7 : Define a function named "matchOrdersBetweenOrderBookAndOrderFile" that takes the order book DataFrame and order file DataFrame as input and performs matching between the two datasets to find closed and unclosed orders.

Inside the "matchOrdersBetweenOrderBookAndOrderFile" function:

a. Extract Buy orders From order Book and Rank the buy orders in the order book on the quantity and order time

Eg:Rank() Over(Partition By quantity order By orderTime Asc)

b. Extract Sell orders From order Book and Rank the buy orders in the order book on the quantity and order time

Eg:Rank() Over(Partition By quantity order By orderTime Asc)

c. Extract Buy orders From order File and Rank the buy orders in the order File on the quantity , order price and order time

Eg:Rank() Over(Partition By quantity order By orderPrice Desc ,orderTime Asc)

d. Extract Sell orders From order File and Rank the buy orders in the order book on the quantity, order price and order time

Eg:Rank() Over(Partition By quantity order By orderPrice Asc ,orderTime Asc)

e. Match the buy orders from the order book and sell orders from order File based on order quantity and rank

f. Match the sell orders from the order book and buy orders from order File based on order quantity and rank

h. Create a DataFrame of closed orders by combining the matched buy and sell orders.

i. Extract the closed Order Ids from the previous step result

j. Filter the order book and order file DataFrames to get unclosed orders by excluding the closed order IDs.

k. Return the closed orders DataFrame and unclosed order DataFrames.

Step-8 : Define a function named "matchOrderWithInTheOrderFile" that takes the order file DataFrame as input and performs matching within the order file dataset to find closed and unclosed orders.

Inside the "matchOrderWithInTheOrderFile" function:

a. Rank the buy orders in the order file DataFrame based on quantity and order time.

b. Rank the sell orders in the order file DataFrame based on quantity and order time.

c. Match the buy and sell orders within the order file based on quantity and rank.

d. Create a DataFrame of closed order IDs by combining the matched buy and sell orders.

e. Filter the order file DataFrame to get unclosed orders by excluding the closed order IDs.

f. Return the closed order DataFrame and unclosed order DataFrame.

Step-9 : Define a function named "matchOrders" that takes the order book DataFrame and order DataFrame as input and performs matching between the two datasets.

Inside the "matchOrders" function:

a. Call the "matchOrdersBetweenOrderBookAndOrderFile" function to get the closed and unclosed orders from the order book and order file datasets.

b. Call the "matchOrderWithInTheOrderFile" function to get additional closed and unclosed orders within the order file dataset.

c. Combine the closed order DataFrames from the order book and order file datasets.

d. Combine the unclosed order DataFrames from the order book and order file datasets.

e. Return the combined closed order DataFrame and combined unclosed order DataFrame.

Step-10: Define a function named "mergeFiles" that takes the source and destination paths

Inside the "mergeFiles" function:

a.check if the source path exists .

b.If the source path exists then merge Files present in the source path to Destination path using FileUtil.copyMerge utility

Step-11 : Define the "main" function that takes command-line arguments as input.

Inside the "main" function:

a. Parse the command-line arguments for file paths and names.

b. Define the schema for the orders DataFrame.

c. Read the orders DataFrame and order book DataFrame using the "readCsvFile" function.

d. Call the "matchOrders" function to get the closed and unclosed orders.

e. Write the unclosed orders and closed orders to temporary files.

f. Archive the original orders file to the specified archival path.

g. Archive the order Book file to the specified archival path.

h. Merge the unclosed Orders file to the order Book path received from the command line argument to create a single orderBook file

i. Merge the closed Orders file to the closedOrdersFilePath received from the command line argument with the given naming pattern.

5.Design Overview:

The code is structured as an object named "TradeExecution" containing various methods for different processing tasks. The main method serves as the entry point for the code. Here is an overview of the design:

• The code starts by creating a Spark session using GetSparkSession.getSparkSession("TradeMatchEngine").

• Configuration settings are adjusted to disable the creation of "\_SUCCESS" files upon job completion.

• An instance of the Hadoop FileSystem is created to interact with the file system, and the checksum verification is disabled.

• The code defines a method readCsvFile that reads a CSV file using the provided file path and schema. It returns a DataFrame containing the file's contents. If the file doesn't exist, an empty DataFrame is created.

• Another method, archiveFile, is implemented to archive a file by renaming it to a specified destination path using the Hadoop FileSystem API.

• Another method, mergeFiles, is implemented to merge multiple files in to a single file to the destination path using the Hadoop FileSystem API.

• The code defines variables for generating timestamps and a matchOdersBetweenOrderBookAndOrderFile method for matching orders between an order book DataFrame and an order file DataFrame. The method returns three DataFrames: matched orders, unclosed orders from the book, and unclosed orders from the file.

• Similarly, a method named matchOrderWithInTheOrderFile matches orders within the order file DataFrame and returns the closed orders and unclosed orders from the file.

• The matchOrders method coordinates the matching process by invoking the previous two methods and returns the closed orders DataFrame and unclosed orders DataFrame.

• In the main method, command-line arguments are retrieved, and file paths and names are initialized accordingly. Schema definitions for orders and trade matches are created.

• CSV files are read using readCsvFile for orders and order book data.

• Matching operations are performed using matchOrders, which returns the closed and unclosed orders.

• The closed orders are written to a CSV file to appended to a temporary file in the output path

• The temporary file will be merged into a single file into the output path provided

• The unclosed orders are appended to a temporary file in the order book path.

• The original order file is archived, and the temporary order book file is renamed to replace the original file.

• Finally, the Spark session is stopped.

### 6. Input and Output:

• Input: The code expects command-line arguments specifying file paths and names for the orders file, order book file, closed orders output file, and order file archival path.

Output: The code produces a closed orders CSV file and appends unclosed orders to a temporary file in the order book path.

Sample Input and Output :

Example 1:

Input :

exampleOrders.csv:

1,Steve,1623239770,SELL,72,1550

2,Ben,1623239771,BUY,8,148

3,Steve,1623239772,SELL,24,6612

4,Kim,1623239773,SELL,98,435

5,Sarah,1623239774,BUY,72,5039

6,Ben,1623239775,SELL,75,6356

7,Kim,1623239776,BUY,38,7957

8,Alex,1623239777,BUY,51,218

9,Jennifer,1623239778,SELL,29,204

10,Alicia,1623239779,BUY,89,7596

11,Alex,1623239780,BUY,70,2351

12,James,1623239781,SELL,89,4352

13,Sarah,1623239782,SELL,98,8302

14,Alicia,1623239783,BUY,56,8771

15,Alex,1623239784,BUY,83,737

16,Andrew,1623239785,SELL,15,61

17,Steve,1623239786,BUY,62,4381

18,Ben,1623239787,BUY,33,5843

19,Alicia,1623239788,BUY,20,5255

20,James,1623239789,SELL,68,4260

output:

closedOrders (tradematch\_20230612182400.csv):

5,1,1623239774,72,1550

12,10,1623239781,89,7596

orderBook (orderbook.csv) :

2,Ben,1623239771,BUY,8,148

3,Steve,1623239772,SELL,24,6612

4,Kim,1623239773,SELL,98,435

6,Ben,1623239775,SELL,75,6356

7,Kim,1623239776,BUY,38,7957

8,Alex,1623239777,BUY,51,218

9,Jennifer,1623239778,SELL,29,204

11,Alex,1623239780,BUY,70,2351

13,Sarah,1623239782,SELL,98,8302

14,Alicia,1623239783,BUY,56,8771

15,Alex,1623239784,BUY,83,737

16,Andrew,1623239785,SELL,15,61

17,Steve,1623239786,BUY,62,4381

18,Ben,1623239787,BUY,33,5843

19,Alicia,1623239788,BUY,20,5255

20,James,1623239789,SELL,68,4260

Example2:

Input :

orderFile (exampleOrders.csv):

21,Jane,1623239790,BUY,98,8312

22,Chris,1623239792,BUY,98,8300

23,Willium,1623239793,SELL,38,8000

24,Christina,1623239794,BUY,100,9000

25,Sravan,1623239795,BUY,38,8500

26,Arun,1623239796,SELL,38,8600

27,Kiran,1623239796,SELL,38,8650

orderBook( orderBook.csv from the example1):

2,Ben,1623239771,BUY,8,148

3,Steve,1623239772,SELL,24,6612

4,Kim,1623239773,SELL,98,435

6,Ben,1623239775,SELL,75,6356

7,Kim,1623239776,BUY,38,7957

8,Alex,1623239777,BUY,51,218

9,Jennifer,1623239778,SELL,29,204

11,Alex,1623239780,BUY,70,2351

13,Sarah,1623239782,SELL,98,8302

14,Alicia,1623239783,BUY,56,8771

15,Alex,1623239784,BUY,83,737

16,Andrew,1623239785,SELL,15,61

17,Steve,1623239786,BUY,62,4381

18,Ben,1623239787,BUY,33,5843

19,Alicia,1623239788,BUY,20,5255

20,James,1623239789,SELL,68,4260

output :

closedOrders (tradematch\_20230612202054.csv) :

23,7,1623239793,38,7957

21,4,1623239790,98,435

22,13,1623239792,98,8302

26,25,1623239796,38,8500

orderBook(orderbook.csv) :

2,Ben,1623239771,BUY,8,148

3,Steve,1623239772,SELL,24,6612

6,Ben,1623239775,SELL,75,6356

8,Alex,1623239777,BUY,51,218

9,Jennifer,1623239778,SELL,29,204

11,Alex,1623239780,BUY,70,2351

14,Alicia,1623239783,BUY,56,8771

15,Alex,1623239784,BUY,83,737

16,Andrew,1623239785,SELL,15,61

17,Steve,1623239786,BUY,62,4381

18,Ben,1623239787,BUY,33,5843

19,Alicia,1623239788,BUY,20,5255

20,James,1623239789,SELL,68,4260

24,Christina,1623239794,BUY,100,9000

27,Kiran,1623239796,SELL,38,8650

### 6. Error Handling:

• If any exception occurs during execution, it is caught in a catch block, and the exception stack trace is printed.

• Finally, the Spark session is stopped to release resources.

Note: The High-Level Design document provides an overview of the code's structure and functionality. For a more detailed understanding, refer to the code comments and review the specific methods and operations performed in the code.