LOGIC BATCH LAYER

We will consider the solution for batch layer is pre-process data steps before starting process streaming data with Kafka in next step.

1. Load card_transactions.csv into MongoDB

Firstly, we need to load data from card_transactions.csv into MongoDB database. (Make sure MongoDB has been correctly installed)

```
aws s3 cp s3://history-transactions/card_transactions.csv - |
mongoimport --db transaction_db --collection
card_transactions --type csv --headerline
```

P/s: If we work on local machine instead of AWS, we can use the command below to load data into MongoDB

```
mongoimport --db transaction_db --collection
card_transactions --type csv --file /Users/nickynguyen/
Downloads/card_transactions.csv --headerline
```

```
[hadoop@ip-172-31-74-203 ~]$ mongosh
Current Mongosh Log ID: 645d819a9891c8d64e68a09d
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB:          6.0.5
Using Mongosh:          1.8.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2023-05-12T00:00:00.266+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-05-12T00:00:00.266+00:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test> use transaction_db
switched to db transaction_db
transaction_db> db.card_transactions.findOne()
{
  _id: ObjectId("645d8194389154a318c0c087"),
  card_id: Long("348702330256514"),
  member_id: Long("37495066290"),
  amount: 4310362,
  postcode: 33946,
  pos_id: Long("614677375609919"),
  transaction_dt: '11-02-2018 00:00:00',
  status: 'GENUINE'
}
```

There are 53292 rows have been loaded successfully.

2. Ingest data from card_member.csv and member_score.csv

We need to connect PySpark with MongoDB before ingesting data.
We will need to go to /etc/mongod.conf, then look for the bindIp option and set it to 0.0.0.0. Use the command below:

```
sudo vi /etc/mongod.conf
```

Then go to the EMR master node security group in AWS console. We will need to add an inbound rule to allow traffic on port 27017 with source as the EMR slave node security group.

After that, connect PySpark by the following configuration, take into account the read and write connection. (The IP have to be set by the Private IPv4 of the Master node)

```
pyspark --conf "spark.mongodb.read.connection.uri=mongodb://
172.31.67.33:27017/transaction_db.card_transactions?
readPreference=primaryPreferred" --conf
"spark.mongodb.write.connection.uri=mongodb://
172.31.67.33:27017/transaction_db.tb_lookup" --packages
org.mongodb.spark:mongo-spark-connector_2.12:10.1.1
```

Start loading data from card_transactions(MongoDB) and card_member.csv and member_score.csv into data frames.

```
df_tran = spark.read.format("mongodb").load()

df_card =
spark.read.options(inferSchema='True',header='True').csv('s3:
//history-transactions/card_member.csv')

df_score =
spark.read.options(inferSchema='True',header='True').csv('s3:
//history-transactions/member_score.csv')

>>> df_tran.printSchema()
root
 |-- _id: string (nullable = true)
 |-- amount: integer (nullable = true)
 |-- card_id: long (nullable = true)
 |-- member_id: long (nullable = true)
 |-- pos_id: long (nullable = true)
 |-- postcode: integer (nullable = true)
 |-- status: string (nullable = true)
```

```
 |-- transaction_dt: string (nullable = true)

>>> df_card.printSchema()
root
 |-- card_id: long (nullable = true)
 |-- member_id: long (nullable = true)
 |-- member_joining_dt: timestamp (nullable = true)
 |-- card_purchase_dt: string (nullable = true)
 |-- country: string (nullable = true)
 |-- city: string (nullable = true)

>>> df_score.printSchema()
root
 |-- member_id: long (nullable = true)
 |-- score: integer (nullable = true)
```

Then create temporary tables based on those data frames

```
>>> df_tran.createOrReplaceTempView('tb_tran')
>>> df_card.createOrReplaceTempView('tb_card')
>>> df_score.createOrReplaceTempView('tb_score')
```

```
>>> df_tran.createOrReplaceTempView('tb_tran')
>>> spark.sql('SELECT * FROM tb_tran LIMIT 3').show()
+--------------------+-------+---------------+-----------+---------------+--------+-------+-------------------+
|                 _id| amount|        card_id|  member_id|         pos_id|postcode| status|     transaction_dt|
+--------------------+-------+---------------+-----------+---------------+--------+-------+-------------------+
|645fd356ab3b87d97...| 330148|348702330256514|37495066290|614677375609919|   33946|GENUINE|11-02-2018 00:00:00|
|645fd356ab3b87d97...|9084849|348702330256514|37495066290|614677375609919|   33946|GENUINE|11-02-2018 00:00:00|
|645fd356ab3b87d97...| 136052|348702330256514|37495066290|614677375609919|   33946|GENUINE|11-02-2018 00:00:00|
+--------------------+-------+---------------+-----------+---------------+--------+-------+-------------------+
```

```
>>> df_card.createOrReplaceTempView('tb_card')
>>> spark.sql('SELECT * FROM tb_card LIMIT 3').show()
+---------------+--------------+-------------------+----------------+-------------+----------+
|        card_id|     member_id|  member_joining_dt|card_purchase_dt|      country|      city|
+---------------+--------------+-------------------+----------------+-------------+----------+
|340028465709212|  9250698176266|2012-02-08 06:04:13|           05/13|United States| Barberton|
|340054675199675|835873341185231|2017-03-10 09:24:44|           03/17|United States|Fort Dodge|
|340082915339645|512969555857346|2014-02-15 06:30:30|           07/14|United States|    Graham|
+---------------+--------------+-------------------+----------------+-------------+----------+

>>> df_score.createOrReplaceTempView('tb_score')
>>> spark.sql('SELECT * FROM tb_score LIMIT 3').show()
+-------------+-----+
|    member_id|score|
+-------------+-----+
|  37495066290|  339|
| 117826301530|  289|
|1147922084344|  393|
+-------------+-----+
```

3. Create and insert data for Look-up table

After all provided data has been successfully loaded into data frame, we will start to create and insert data for look-up table.

Use the following command to create Look-up table:

```
spark.sql("CREATE TABLE tb_lookup (card_id STRING, ucl
DOUBLE, postcode STRING, transaction_dt STRING, score INT)")
```

Firstly, we will take last 10 transactions for each card from table tb_tran

```
df_10trans = spark.sql("\
SELECT card_id, amount, postcode, transaction_dt, status, rn \
FROM (\
SELECT card_id, amount, postcode, transaction_dt, status,
ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY
unix_timestamp(transaction_dt,'dd-MM-yyyy hh:mm:ss') DESC) AS rn \
FROM tb_tran \
WHERE status = 'GENUINE') a \
WHERE a.rn <= 10”)
```

Then convert it to table tb_10trans

```
>>> df_10trans.createOrReplaceTempView('tb_10trans')
>>> spark.sql('SELECT * FROM tb_10trans LIMIT 20’).show()
```

```
>>> spark.sql('SELECT * FROM tb_10trans LIMIT 20').show()
+---------------+-------+--------+-------------------+-------+---+
|        card_id| amount|postcode|     transaction_dt| status| rn|
+---------------+-------+--------+-------------------+-------+---+
|340028465709212|8696557|   24658|02-01-2018 03:25:35|GENUINE|  1|
|340028465709212| 430409|   58270|15-11-2017 01:59:54|GENUINE|  2|
|340028465709212|6503191|   84776|09-11-2017 07:18:21|GENUINE|  3|
|340028465709212|8884049|   25537|07-10-2017 09:17:12|GENUINE|  4|
|340028465709212|9291309|   31322|12-08-2017 08:29:54|GENUINE|  5|
|340028465709212|8370505|   84056|12-07-2017 02:51:29|GENUINE|  6|
|340028465709212|9687739|   51542|05-07-2017 11:05:55|GENUINE|  7|
|340028465709212|6500086|   25040|24-06-2017 01:13:31|GENUINE|  8|
|340028465709212| 581323|   46182|17-05-2017 12:36:12|GENUINE|  9|
|340028465709212|5118701|   12045|30-03-2017 04:09:10|GENUINE| 10|
|340054675199675|  29445|   50140|15-01-2018 10:56:43|GENUINE|  1|
|340054675199675|9728785|   77373|10-01-2018 02:47:11|GENUINE|  2|
|340054675199675|2223104|   35973|09-01-2018 10:59:10|GENUINE|  3|
|340054675199675|1201277|   84530|28-12-2017 05:48:04|GENUINE|  4|
|340054675199675|6140357|   40023|18-12-2017 10:33:04|GENUINE|  5|
|340054675199675|7914699|   41844|12-12-2017 07:04:51|GENUINE|  6|
|340054675199675|7573707|   12024|06-12-2017 08:52:38|GENUINE|  7|
|340054675199675|2797924|   54141|04-12-2017 12:59:15|GENUINE|  8|
|340054675199675|7876899|   71047|27-11-2017 01:54:59|GENUINE|  9|
|340054675199675|5418389|   21084|05-11-2017 12:00:53|GENUINE| 10|
+---------------+-------+--------+-------------------+-------+---+
```

Next, we will calculate UCL for each card

```
df_ucl = spark.sql("\
SELECT a.card_id, (a.avge + (3 * a.std)) as UCL \
FROM (\
SELECT t.card_id, AVG(t.amount) AS avge, STDDEV(t.amount) as std \
FROM tb_10trans t \
GROUP BY t.card_id) a")
```

Then convert it to table tb_ucl

```
>>> df_ucl.createOrReplaceTempView('tb_ucl')
>>> spark.sql('SELECT * FROM tb_ucl LIMIT 3').show()
```

```
>>> df_ucl.createOrReplaceTempView('tb_ucl')
>>> spark.sql('SELECT * FROM tb_ucl LIMIT 3').show()
+--------------+--------------------+
|       card_id|                 UCL|
+--------------+--------------------+
|340028465709212|1.6685076623853374E7|
|340054675199675|1.5032693399975928E7|
|340082915339645|1.5323729774843596E7|
+--------------+--------------------+
```

Finally, join those 2 tables with tb_card and tb_score to insert data into look-up table

```
spark.sql("INSERT INTO TABLE tb_lookup \
SELECT trans.card_id, ucl.ucl, trans.postcode,
trans.transaction_dt, cdsc.score \
FROM  tb_10trans trans \
JOIN tb_ucl ucl \
ON ucl.card_id = trans.card_id \
JOIN (\
SELECT DISTINCT crd.card_id, scr.score \
FROM tb_card crd \
JOIN tb_score scr \
ON crd.member_id = scr.member_id) AS cdsc \
ON trans.card_id = cdsc.card_id \
WHERE trans.rn = 1")

>>> spark.sql('SELECT * FROM tb_lookup LIMIT 3').show()
```

```
>>> spark.sql('SELECT * FROM tb_lookup LIMIT 3').show()
+---------------+--------------------+--------+-------------------+-----+
|        card_id|                 ucl|postcode|     transaction_dt|score|
+---------------+--------------------+--------+-------------------+-----+
|5411141346922507|1.5997512196104523E7|   25156|20-09-2017 09:59:16|  225|
|4502008970767222| 1.684959540133459E7|   93010|31-01-2018 09:39:57|  657|
|5341963603599990| 1.516390152866365E7|   18419|09-01-2018 07:28:30|  554|
+---------------+--------------------+--------+-------------------+-----+
```

Finally, save the lookup table into MongoDB

```
df_lookup.write.format("mongodb").mode("append").save()
```

```
switched to db transaction_db
transaction_db> db.card_transactions.findOne()
{
  _id: ObjectId("646a94c87760a83f9571a002"),
  card_id: Long("348702330256514"),
  member_id: Long("37495066290"),
  amount: 4310362,
  postcode: 33946,
  pos_id: Long("614677375609919"),
  transaction_dt: '11-02-2018 00:00:00',
  status: 'GENUINE'
}
transaction_db> db.tb_lookup.findOne()
{
  _id: ObjectId("646a9b313ec71934249ddea4"),
  card_id: Long("5411141346922507"),
  ucl: 15997512.196104523,
  postcode: 25156,
  transaction_dt: '20-09-2017 09:59:16',
  score: 225
}
```

Now, in MongoDB database, we have 2 tables card_transactions and tb_lookup are ready for the next streaming part.