BATCH LAYER LOGIC

In the next step, we will process streaming data with Kafka. However, before we get to that, we will first consider the solution for the batch layer, which involves pre-processing of the data.

1. Load card_transactions.csv into MongoDB

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

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

```
[hadoop@ip-172-31-74-8 ~]$ mongosh
Current Mongosh Log ID: 65c4811dd77c3b2b3a029a9d
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.3
Using MongoDB:          7.0.5
Using Mongosh:          2.1.3

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
   2024-02-08T06:58:30.366+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-02-08T06:58:30.366+00:00: vm.max_map_count is too low
------

test> use card_transactions_db
switched to db card_transactions_db
card_transactions_db> db.card_transactions.findOne()
null
card_transactions_db> use transaction_db
switched to db transaction_db
transaction_db> db.card_transactions.findOne()
{
  _id: ObjectId('65c47d6b2bf0e94c333ab569'),
  card_id: Long('348702330256514'),
  member_id: Long('374950662290'),
  amount: 4310362,
  postcode: 33946,
  pos_id: Long('614677375609919'),
  transaction_dt: '11-02-2018 00:00:00',
  status: 'GENUINE'
}
transaction_db>

transaction_db>

transaction_db>

transaction_db>
```

```
~
~
~
[hadoop@ip-172-31-74-8 ~]$ aws s3 cp s3://creditcardcapstone/card_transactions.csv - | mongoimport --db transaction_db --collection card_transactions --type csv --headerline
2024-02-08T07:06:19.429+0000    connected to: mongodb://localhost/
2024-02-08T07:06:20.574+0000    53292 document(s) imported successfully. 0 document(s) failed to import.
[hadoop@ip-172-31-74-8 ~]$
```

There are 53292 rows have been loaded successfully.

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

- Before we can ingest data, it's necessary to establish a connection between PySpark and MongoDB. To do this, we'll need to modify the /etc/mongod.conf file. Specifically, we'll need to find the bind1p option and change its value to 0.0.0.0. The command below can be used for this purpose.

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

- Next, navigate to the security group for the EMR master node in the AWS console. It's necessary to add an inbound rule that permits traffic on port 27017, with the source being the security group for the EMR slave node. This step is also crucial.

- Subsequently, establish a connection with PySpark using the provided configuration. It's important to note that this configuration should account for both read and write connections. The IP address should be set to 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.csvand member_score.csv into data frames.

```
crd=spark.read.options(inferSchema='True',header='True').csv('s3://creditcard
    capstone/card_member.csv')
mem= spark.read.options(inferSchema='True',header='True').csv('s3://credit
    cardcapstone/member_score.csv')
```

```
>>> crd.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)
```

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

- *Then create temporary tables based on those data frames*

```
>>> crd.createOrReplaceTempView('card_mem')
>>> mem.createOrReplaceTempView('mem_score')
```

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

3. Create and insert data for Look-up table

- *To create the look-up table, please use the command provided below.*

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

- *Initially, we will retrieve the most recent 10 transactions for each card from the transact_table.*

```
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 transact_table \
```

```
WHERE status = 'GENUINE') a \
WHERE a.rn <= 10")
```

- *Next, we will transform this data into a new table named tb_10transact*

```
>>> df_10trans.createOrReplaceTempView('table_10transact')
>>> spark.sql('SELECT * FROM table_10transact LIMIT 20').show()
```

```
>>> spark.sql('SELECT * FROM table_10transact 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_10transact t \
GROUP BY t.card_id)  a")
```

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

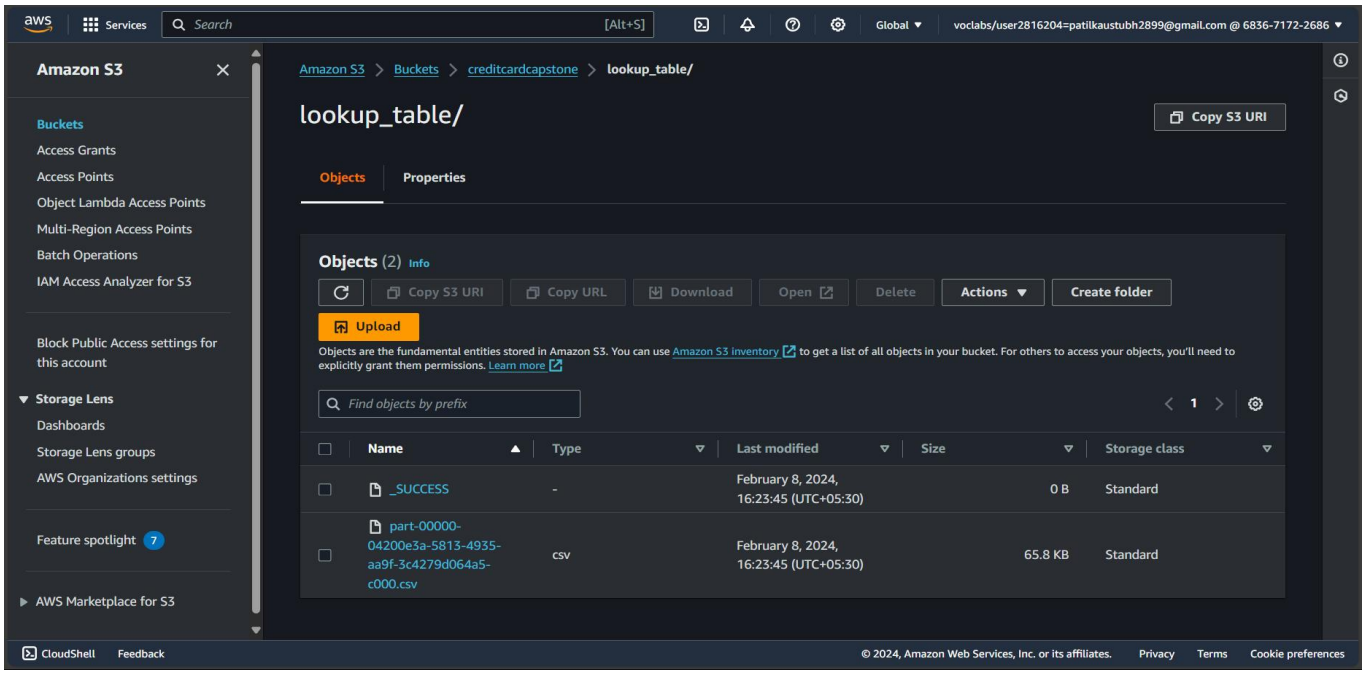
```
>>> spark.sql("INSERT INTO TABLE lookup_table \
... SELECT trans.card_id, ucl.ucl, trans.postcode, trans.transaction_dt, CAST(cdsc.score as double)\
... FROM table_10transact trans \
... JOIN UCL_table ucl \
... ON ucl.card_id = trans.card_id \
... JOIN (\
... SELECT DISTINCT crd.card_id, scr.score \
... FROM card_mem crd \
... JOIN mem_score scr \
... ON crd.member_id = scr.member_id) AS cdsc \
... ON trans.card_id = cdsc.card_id \
... WHERE trans.rn = 1")
DataFrame[]
>>> spark.sql('SELECT * FROM lookup_table LIMIT 3').show()
+---------------+-------------------+--------+-------------------+-----+
|        card_id|                ucl|postcode|     transaction_dt|score|
+---------------+-------------------+--------+-------------------+-----+
|340028465709212|1.6685076623853374E7|   24658|02-01-2018 03:25:35|  233|
|340054675199675|1.5032693399975928E7|   50140|15-01-2018 10:56:43|  631|
|340082915339645|1.5323729774843596E7|   41754|03-11-2017 09:06:10|  407|
+---------------+-------------------+--------+-------------------+-----+

>>> 
```

Finally, save the lookup table into S3 bucket

```
CODE :
>> lookup_df=spark.sql("SELECT * FROM lookup_table")
>> lookup_df.coalesce(1).write.format('csv').options(header='True'
   delimiter=',').mode("overwrite").save("s3://creditcardcapstone/looku
   p_table")
```

```
>>> lookup_df = sqlContext.sql("SELECT * FROM lookup_table")
>>> lookup_df.coalesce(1).write.format('csv').options(header='True',delimiter=',').mode('overwrite').save("s3://creditcardcapstone/lookup_table")
```

Lookup table successfully uploaded on s3 bucket



THE END

----------X-X-X----------