Transaction Analysis Script for Hadoop and NoSQL

1. Retrieve the most recent 10 transactions for each card.

- *First step: Extract the latest 10 transactions for each card from tablee_10transact.*

```
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 table_10transact  \
WHERE status = 'GENUINE') a \
WHERE a.rn <= 10")
```

- *Next, transform the data into the tb_10trans table.*

```
>>> 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|
+---------------+-------+--------+-------------------+-------+---+

>>>
```

2. Calculate UCL

- *Following that, we'll compute the Upper Control Limit (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 table_10transact t \
GROUP BY t.card_id) a")
```

- *Then, the data is transformed into the UCL_table table.*

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

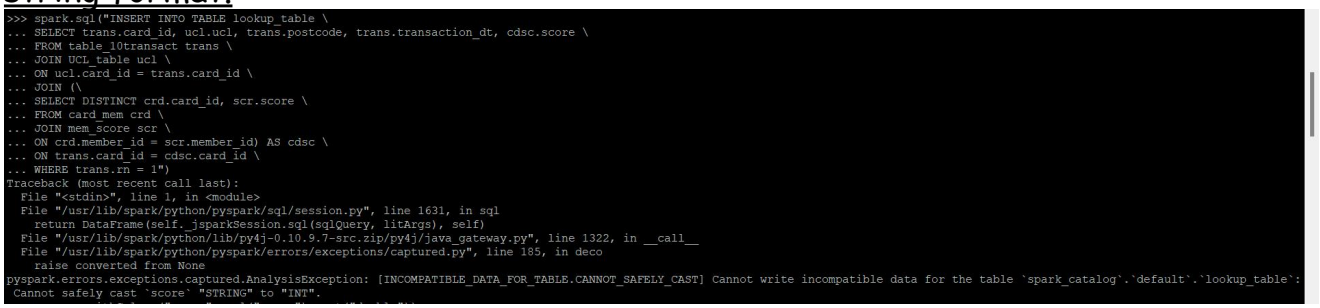
```
>>> df_ucl = spark.sql("\
... SELECT a.card_id, (a.avge + (3 * a.std)) as UCL \
... FROM (\
... SELECT t.card_id, AVG(t.amount) AS avge, STDDEV(t.amount) as std \
... FROM table_10transact t \
... GROUP BY t.card_id) a")
>>> df_ucl.createOrReplaceTempView('UCL_table')
>>> spark.sql('SELECT * FROM UCL_table LIMIT 3').show()
+---------------+--------------------+
|        card_id|                 UCL|
+---------------+--------------------+
|340028465709212|1.6685076623853374E7|
|340054675199675|1.5032693399975928E7|
|340082915339645|1.5323729774843596E7|
+---------------+--------------------+

>>>
```

3. Insert data for look-up table

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

```
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")
```

Here main reason for type casting the score was by default the score was in the string format.



```
>>> spark.sql("INSERT INTO TABLE lookup_table \
... SELECT trans.card_id, ucl.ucl, trans.postcode, trans.transaction_dt, cdsc.score \
... 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")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/spark/python/pyspark/sql/session.py", line 1631, in sql
    return DataFrame(self._jsparkSession.sql(sqlQuery, litArgs), self)
  File "/usr/lib/spark/python/lib/py4j-0.10.9.7-src.zip/py4j/java_gateway.py", line 1322, in __call__
  File "/usr/lib/spark/python/pyspark/errors/exceptions/captured.py", line 185, in deco
    raise converted from None
pyspark.errors.exceptions.captured.AnalysisException: [INCOMPATIBLE_DATA_FOR_TABLE.CANNOT_SAFELY_CAST] Cannot write incompatible data for the table `spark_catalog`.`default`.`lookup_table`:
 Cannot safely cast `score` "STRING" to "INT".
```

>>> 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|
+---------------+--------------------+--------+-------------------+-----+

>>>
```

4. Saving lookup table

- *Save the lookup table into MongoDB*

```
lookup_df = sqlContext.sql("SELECT * FROM lookup_table")
lookup_df.coalesce(1).write.format('csv').options(header='Tru
e',delimiter=',').mode('overwrite').save("s3://creditcardcaps
tone/lookup_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")
```

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