Logic of Credit Card Fraud Detection Project – Mid Submission

I have submitted below artifacts as part of Credit Card Fraud Detection Project – Mid Submission.

1. LoadCreateNoSQL.pdf
2. DataIngestion.pdf
3. CreateNoSQL.pdf
4. PreAnalysis.pdf
5. ScriptExecution.pdf
6. LogicMid.pdf

Explanation of the solution to the batch layer problem

Credit Card Fraud Detection Problem Statement

You are supposed to build solution to cater to Credit Card Fraud detection. This will detect fraudelent transactions, when card member swipes their card for payment, the transaction is classified as fraudulent or authentic based on a set of predefined rules. If fraud is detected, then the transaction must be declined. Incorrectly classifying a transaction as fraudulent will incur huge losses to the company and also provoke negative consumer sentiment.

As part of this, the relevant information about the customers needs to be continuously updated on a platform from where the customer support team can retrieve relevant information in real-time to resolve customer complaints and queries.

As part of the project, broadly, below are the tasks for mid submission.

Task 1: Load the transactions history data (card_transactions.csv) in a NoSQL database.
Task 2: Ingest the relevant data from AWS RDS to Hadoop.
Task 3: Create a look-up table with columns specified earlier in the problem statement.
Task 4: After creating the table, you need to load the relevant data in the lookup table.

The following tables containing data will be created to solve this problem:

card_member – This table will store **the cardholder's data in a central AWS RDS**

- card_id: This refers to the card number.
- member_id: This is the 15-digit member ID of the cardholder.

- member_joining_dt: This is the date and time of joining of new member.
- card_purchase_dt: This is the date on which the card was purchased.
- country: This is the country in which the card was purchased.
- city: This is the city in which the card was purchased.

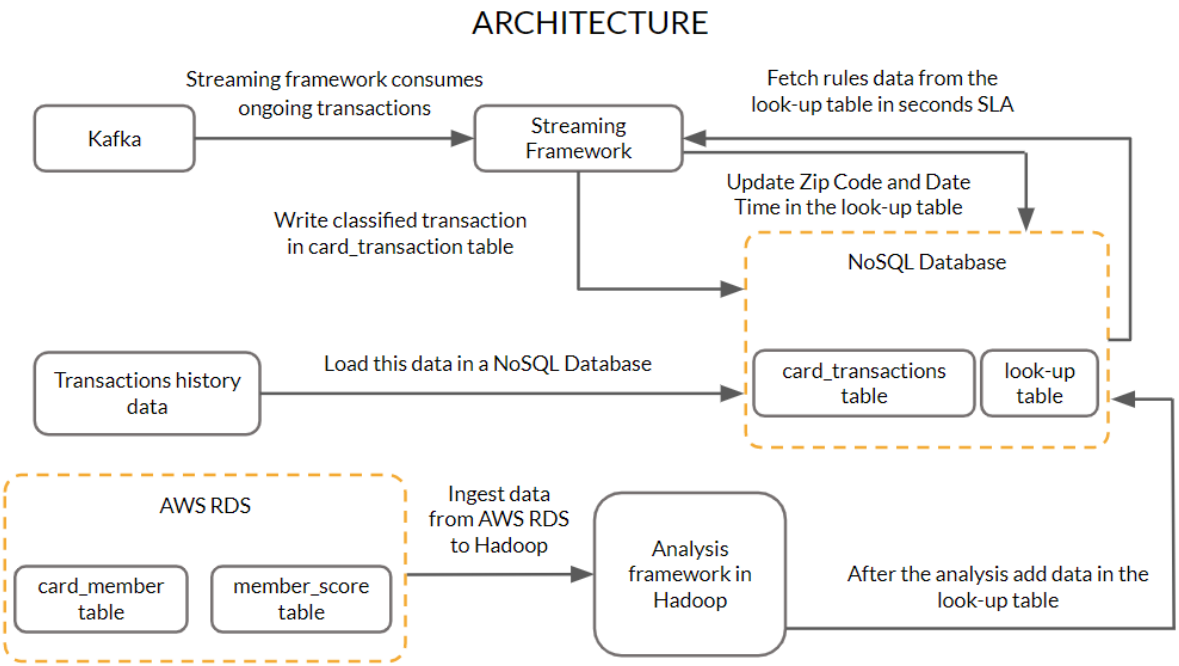
card_transactions – This table will store all **incoming transactions (fraud/genuine)** swiped at point of sale (POS) terminals.

- card_id: This refers to the card number.
- member_id: This is the 15-digit member ID of the cardholder.
- amount: This is the amount that is swiped with respect to the card_id.
- postcode: This is the ZIP code at which this card was swiped (marking the location of an event).
- pos_id: This is the merchant's POS terminal ID, using which the card was swiped.
- transaction_dt: This is the date and time of the transaction.
- status: This indicates whether the transaction was approved or not, with a genuine/fraud value.

member_score – This table will store the member credit score data in a central AWS RDS.

- member_id: This is the 15-digit member ID of the cardholder.
- score: This is the score assigned to a member defining their credit history, generated by upstream systems.

Architecture and Approach



Explanation of the Batch Layer

The Batch Layer will load required historical credit card transaction data and master data such as card_member and member_scre table into NOSQL database ie Hbase. Based on the rules provided 3 parameters such as a) **Upper Control Limit (UCL)** b) **credit score** and c) **zip code of distance** are derived for each card member and lookup table is populated. This data will be used to determine if the credit card transactions are fraudulent or authentic.

The details of the member and the credit score associated with members are hosted on a central AWS RDS server.

The historical transaction data will be provided as a CSV file. You need to use appropriate ingestion methods available to bring the card_member and member_score data from the AWS RDS into a Hadoop platform.

You also need to load the historical card transactions into a NoSQL database. This data is then processed to fill data in the look-up table.

The card_transactions table also needs to be updated with all the details along with the classification of the transactions.

The **lookup table** will contain the following details:

- Card id
- Upper control limit (UCL)
- Postcode of the last transaction
- Transaction date of the last transaction
- The credit score of the member

The implementation is 4 tasks is provided as below.

1. **Task 1:** Load the transactions history data (card_transactions.csv) in a NoSQL database

   The historical transaction data will be provided as a CSV file. This file will be downloaded into local system, then copied into ec2 instance file system using filezilla. Then this file will be copied to a Hadoop filesystem location which will be loaded into NOSQL database thru a create table script reference to csv file in hadoop file system.

   Hive Scripts

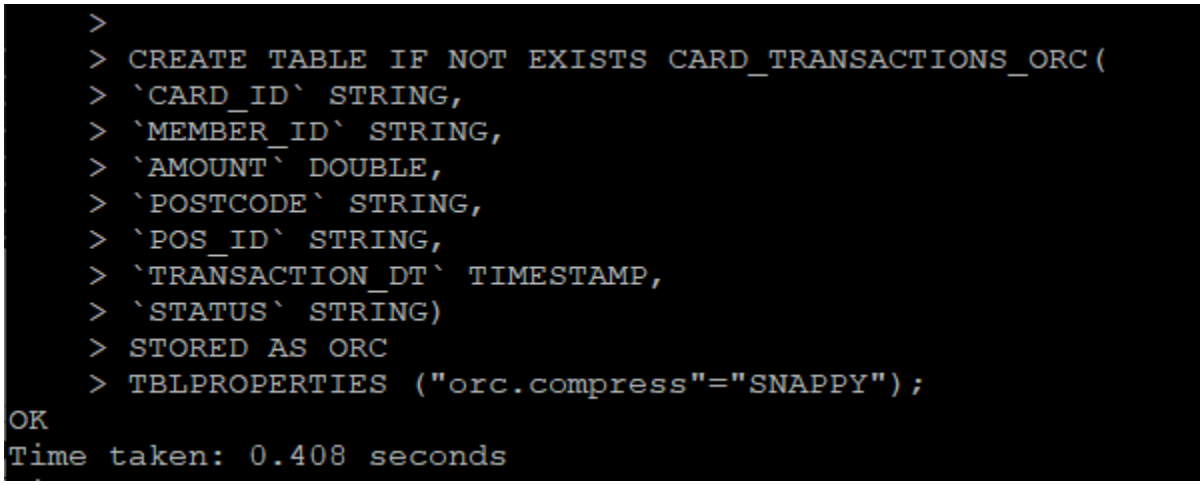
   1. Create external table card_transactions_ext table which will point to HDFS location created earlier.

```
CREATE EXTERNAL TABLE IF NOT EXISTS CARD_TRANSACTIONS_EXT(
`CARD_ID` STRING,
`MEMBER_ID` STRING,
`AMOUNT` DOUBLE,
`POSTCODE` STRING,
`POS_ID` STRING,
`TRANSACTION_DT` STRING,
`STATUS` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/capstone_project/card_transactions'
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
hive>
    > CREATE EXTERNAL TABLE IF NOT EXISTS CARD_TRANSACTIONS_EXT(
    > `CARD_ID` STRING,
    > `MEMBER_ID` STRING,
    > `AMOUNT` DOUBLE,
    > `POSTCODE` STRING,
    > `POS_ID` STRING,
    > `TRANSACTION_DT` STRING,
    > `STATUS` STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LOCATION '/capstone_project/card_transactions'
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.441 seconds
```

2. Create table card_transactions_orc. ORC format will help in better performance.

```
CREATE TABLE IF NOT EXISTS CARD_TRANSACTIONS_ORC(
`CARD_ID` STRING,
`MEMBER_ID` STRING,
`AMOUNT` DOUBLE,
`POSTCODE` STRING,
`POS_ID` STRING,
`TRANSACTION_DT` TIMESTAMP,
`STATUS` STRING)
STORED AS ORC
TBLPROPERTIES ("orc.compress"="SNAPPY");
```



3. Load data in card_transactions_orc while casting timestamp format for transaction_dt column.

```
INSERT OVERWRITE TABLE CARD_TRANSACTIONS_ORC
SELECT CARD_ID, MEMBER_ID, AMOUNT, POSTCODE, POS_ID,
CAST(FROM_UNIXTIME(UNIX_TIMESTAMP(TRANSACTION_DT,'dd-MM-yyyy HH:mm:ss'))
AS TIMESTAMP), STATUS
FROM CARD_TRANSACTIONS_EXT;
```

```
hive>
    > INSERT OVERWRITE TABLE CARD_TRANSACTIONS_ORC
    > SELECT CARD_ID, MEMBER_ID, AMOUNT, POSTCODE, POS_ID, CAST(FROM_UNIXTIME(UNIX_TIMESTAMP(TRANSACTION_DT,'dd-MM-yyyy
 HH:mm:ss')) AS TIMESTAMP), STATUS
    > FROM CARD_TRANSACTIONS_EXT;
Query ID = hdfs_20230518065643_3fdbdfb5-2d1a-4164-9a3d-bb08d432cfa9
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1684390665990_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 6.12 s
--------------------------------------------------------------------------------
Loading data to table capstone_project.card_transactions_orc
OK
Time taken: 16.33 seconds
```

4. Verify transaction_dt and year in card_transactions_orc table.
select year(transaction_dt), transaction_dt from card_transactions_orc limit 10;

```
hive>
    > select year(transaction_dt), transaction_dt from card_transactions_orc limit 10;
OK
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
2018    2018-02-11 00:00:00
Time taken: 0.236 seconds, Fetched: 10 row(s)
hive>
```

2. **Task 2:** The details of the member and the credit score associated with members are hosted on a central AWS RDS server. These 2 table data will be loaded into NOSQL database using sqoop import commands.

Sqoop command used for importing table from RDS to HDFS

a. Run below Sqoop command to import member_score table from RDS into HDFS, from command prompt.

sqoop import --connect jdbc:mysql://upgradawsrds1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table member_score --null-string 'NA' --null-non-string '\\N' --delete-target-dir --target-dir '/capstone_project/member_score' -m 1

```
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
23/05/21 07:21:43 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/redshift/jdbc/redshift-jdbc42-1.2.37.1061.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
23/05/21 07:21:44 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
23/05/21 07:21:44 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
23/05/21 07:21:44 INFO tool.CodeGenTool: Beginning code generation
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatical
ly registered via the SPI and manual loading of the driver class is generally unnecessary.
23/05/21 07:21:45 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `member_score` AS t LIMIT 1
23/05/21 07:21:45 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `member_score` AS t LIMIT 1
23/05/21 07:21:45 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-hadoop/compile/d8ce8213316f36df6c5fa5f56a4c7384/member_score.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
23/05/21 07:21:49 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/d8ce8213316f36df6c5fa5f56a4c7384/member_score
.jar
23/05/21 07:21:51 INFO tool.ImportTool: Destination directory /capstone_project/member_score is not present, hence not deleting.
23/05/21 07:21:51 WARN manager.MySQLManager: It looks like you are importing from mysql.
23/05/21 07:21:51 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
23/05/21 07:21:51 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
23/05/21 07:21:51 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
23/05/21 07:21:51 INFO mapreduce.ImportJobBase: Beginning import of member_score
23/05/21 07:21:51 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
23/05/21 07:21:51 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
23/05/21 07:21:52 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-92-66.ec2.internal/172.31.92.66:8032
23/05/21 07:21:55 INFO db.DBInputFormat: Using read commited transaction isolation
23/05/21 07:21:55 INFO mapreduce.JobSubmitter: number of splits:1
23/05/21 07:21:55 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684652299850_0002
23/05/21 07:21:56 INFO impl.YarnClientImpl: Submitted application application_1684652299850_0002
23/05/21 07:21:56 INFO mapreduce.Job: The url to track the job: http://ip-172-31-92-66.ec2.internal:20888/proxy/application_168465229985
0_0002/
23/05/21 07:21:56 INFO mapreduce.Job: Running job: job_1684652299850_0002
23/05/21 07:22:04 INFO mapreduce.Job: Job job_1684652299850_0002 running in uber mode : false
23/05/21 07:22:04 INFO mapreduce.Job:  map 0% reduce 0%
23/05/21 07:22:11 INFO mapreduce.Job:  map 100% reduce 0%
23/05/21 07:22:11 INFO mapreduce.Job: Job job_1684652299850_0002 completed successfully
23/05/21 07:22:11 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=189631
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=19980
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=213456
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=4447
                Total vcore-milliseconds taken by all map tasks=4447
                Total megabyte-milliseconds taken by all map tasks=6830592
        Map-Reduce Framework
                Map input records=999
                Map output records=999
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=75
                CPU time spent (ms)=2170
                Physical memory (bytes) snapshot=266850304
                Virtual memory (bytes) snapshot=3281862656
                Total committed heap usage (bytes)=242745344
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=19980
23/05/21 07:22:11 INFO mapreduce.ImportJobBase: Transferred 19.5117 KB in 19.6785 seconds (1,015.3227 bytes/sec)
23/05/21 07:22:11 INFO mapreduce.ImportJobBase: Retrieved 999 records.
```

b. Run below Sqoop command to import card_member table from RDS into HDFS, from command prompt.

sqoop import --connect jdbc:mysql://upgradawsrds1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table member_score --null-string 'NA' --null-non-string '\\N' --delete-target-dir --target-dir '/capstone_project/card_member' -m 1

```
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
23/05/21 07:28:19 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/redshift/jdbc/redshift-jdbc42-1.2.37.1061.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
23/05/21 07:28:19 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
23/05/21 07:28:19 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
23/05/21 07:28:19 INFO tool.CodeGenTool: Beginning code generation
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatical
ly registered via the SPI and manual loading of the driver class is generally unnecessary.
23/05/21 07:28:19 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `member_score` AS t LIMIT 1
23/05/21 07:28:20 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `member_score` AS t LIMIT 1
23/05/21 07:28:20 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-hadoop/compile/93e78845b87762a455efc0b8dd316de7/member_score.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
23/05/21 07:28:22 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/93e78845b87762a455efc0b8dd316de7/member_score
.jar
23/05/21 07:28:24 INFO tool.ImportTool: Destination directory /capstone_project/card_member is not present, hence not deleting.
23/05/21 07:28:24 WARN manager.MySQLManager: It looks like you are importing from mysql.
23/05/21 07:28:24 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
23/05/21 07:28:24 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
23/05/21 07:28:24 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
23/05/21 07:28:24 INFO mapreduce.ImportJobBase: Beginning import of member_score
23/05/21 07:28:24 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
23/05/21 07:28:24 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
23/05/21 07:28:24 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-92-66.ec2.internal/172.31.92.66:8032
23/05/21 07:28:26 INFO db.DBInputFormat: Using read commited transaction isolation
23/05/21 07:28:26 INFO mapreduce.JobSubmitter: number of splits:1
23/05/21 07:28:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684652299850_0003
23/05/21 07:28:27 INFO impl.YarnClientImpl: Submitted application application_1684652299850_0003
23/05/21 07:28:27 INFO mapreduce.Job: The url to track the job: http://ip-172-31-92-66.ec2.internal:20888/proxy/application_168465229985
0_0003/
23/05/21 07:28:27 INFO mapreduce.Job: Running job: job_1684652299850_0003
23/05/21 07:28:35 INFO mapreduce.Job: Job job_1684652299850_0003 running in uber mode : false
23/05/21 07:28:35 INFO mapreduce.Job:  map 0% reduce 0%
23/05/21 07:28:41 INFO mapreduce.Job:  map 100% reduce 0%
23/05/21 07:28:41 INFO mapreduce.Job: Job job_1684652299850_0003 completed successfully
23/05/21 07:28:41 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=189630
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=19980
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=174816
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=3642
                Total vcore-milliseconds taken by all map tasks=3642
                Total megabyte-milliseconds taken by all map tasks=5594112
        Map-Reduce Framework
                Map input records=999
                Map output records=999
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=74
                CPU time spent (ms)=2160
                Physical memory (bytes) snapshot=269864960
                Virtual memory (bytes) snapshot=3286827008
                Total committed heap usage (bytes)=244842496
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=19980
23/05/21 07:28:41 INFO mapreduce.ImportJobBase: Transferred 19.5117 KB in 17.3427 seconds (1.1251 KB/sec)
23/05/21 07:28:41 INFO mapreduce.ImportJobBase: Retrieved 999 records.
```

3. **Task 3:** Create a look-up table with columns specified earlier in the problem statement.

   Command to create the Lookup Table

```
CREATE TABLE LOOKUP_DATA_HBASE
(`CARD_ID` STRING,
`UCL` DOUBLE,
`SCORE` INT,
`POSTCODE` STRING,
`TRANSACTION_DT` TIMESTAMP)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping"=":key, lookup_card_family:ucl,
lookup_card_family:score, lookup_transaction_family:postcode,
lookup_transaction_family:transaction_dt")
TBLPROPERTIES ("hbase.table.name" = "lookup_data_hive");
```

   a. In HBase, alter lookup_data_hive table and set VERSIONS to 10 for lookup_transaction_family. We are supposed to store last 10 transactions in lookup table so altering VERSIONS to 10. I have created 2 column families in lookup table namely lookup_card_family and lookup_transaction_family. Column family lookup_card_family has score and ucl as columns and will store only 1 VERSION. Column family lookup_transaction_family has postcode and transaction_dt and will store 10 VERSIONS .

      alter 'lookup_data_hive', {NAME => 'lookup_transaction_family', VERSIONS => 10}

   b. In HBase, check details of lookup_data_hive and confirm that VERSIONS is set to 10 for lookup_transaction_family.

      describe 'lookup_data_hive'

Below 3 parameters are defined by the rules required to determine the authenticity of transactions.

   c. **Upper Control Limit (UCL)** : This is the upper limit on the amount per transaction. This is different from maximum transaction limit on each card. This is indicator of transaction pattern associated with the customer. Upper Control Limit (UCL) is defined as below formula which will be used to derive UCL value for each card_id.

UCL = Moving average + 3 * Standard deviation

The moving average and the standard deviation for each card_id are calculated based on the last 10 amounts credited that were classified as genuine.

d. **Credit score of each member:** This is a straightforward rule, where a member_score table in which member IDs and their respective scores are available. If score is less than 200, that member transaction will be rejected, as they could be defaulter.

e. **Zip Code of distance :** The whole purpose of this rule is to keep a check on the distance between the card owner's current and last transaction location with respect to time. If the distance between the current transaction and the last transaction location with respect to time is greater than a particular threshold, then this raises suspicion on the authenticity of the transaction.

Scripts to implement this.

Hive Commands

f. Create table ranked_card_transactions_orc to store last 10 transactions for each card_id. ORC format will help in better performance.

```
CREATE TABLE IF NOT EXISTS RANKED_CARD_TRANSACTIONS_ORC(
`CARD_ID` STRING,
`AMOUNT` DOUBLE,
`POSTCODE` STRING,
`TRANSACTION_DT` TIMESTAMP,
`RANK` INT)
STORED AS ORC
TBLPROPERTIES ("orc.compress"="SNAPPY");
```

g. Create table card_ucl_orc to store UCL values for each card_id. ORC format will help in better performance.

```
CREATE TABLE IF NOT EXISTS CARD_UCL_ORC(
`CARD_ID` STRING,
`UCL` DOUBLE)
STORED AS ORC
TBLPROPERTIES ("orc.compress"="SNAPPY");
```

h. Load data in ranked_card_transactions_orc table. Here for each card id get top 10 transactions based on the amount column. This is done with SQL using Rank() function partition by card_id sorted by amount in descrending order with max # of transactions <= 10.

```
INSERT OVERWRITE TABLE RANKED_CARD_TRANSACTIONS_ORC
SELECT B.CARD_ID, B.AMOUNT, B.POSTCODE, B.TRANSACTION_DT, B.RANK FROM
(SELECT A.CARD_ID, A.AMOUNT, A.POSTCODE, A.TRANSACTION_DT, RANK()
OVER(PARTITION BY A.CARD_ID ORDER BY A.TRANSACTION_DT DESC, AMOUNT
DESC) AS RANK FROM
(SELECT CARD_ID, AMOUNT, POSTCODE, TRANSACTION_DT FROM
CARD_TRANSACTIONS_HBASE WHERE
STATUS = 'GENUINE') A ) B WHERE B.RANK <= 10;
```

i. Load data in card_ucl_orc table. In innermost query, select card_id, average of amount and standard deviation of amount from card_transactions_orc. In outermost query, select card_id and compute UCL using average and standard deviation with formula (avg + (3 * stddev)). Insert all this data in card_ucl_orc.

```
INSERT OVERWRITE TABLE CARD_UCL_ORC
SELECT A.CARD_ID, (A.AVERAGE + (3 * A.STANDARD_DEVIATION)) AS UCL FROM (
SELECT CARD_ID, AVG(AMOUNT) AS AVERAGE, STDDEV(AMOUNT) AS
STANDARD_DEVIATION FROM
RANKED_CARD_TRANSACTIONS_ORC
GROUP BY CARD_ID) A;
```

Task 4: After creating the table, load the relevant data in the lookup table.

1. Load data in lookup_data_hbase table. Create intermediate table or sort of inline view which can be used in JOIN condition by selecting card_id, score from card_member_orc joining member_score_orc on member_id and name it as CMS. In main query, select card_id, UCL, score, postcode, transaction_dt from ranked_card_transactions_orc joining card_ucl_orc on card_id column and joining cms on card_id where rank is 1. This will ensure that we have obtained data of latest transaction for each card_id.

```
INSERT OVERWRITE TABLE LOOKUP_DATA_HBASE
SELECT RCTO.CARD_ID, CUO.UCL, CMS.SCORE, RCTO.POSTCODE, RCTO.TRANSACTION_DT
FROM RANKED_CARD_TRANSACTIONS_ORC RCTO
JOIN CARD_UCL_ORC CUO
ON CUO.CARD_ID = RCTO.CARD_ID
JOIN (
SELECT DISTINCT CARD.CARD_ID, SCORE.SCORE
FROM CARD_MEMBER_ORC CARD
JOIN MEMBER_SCORE_ORC SCORE
ON CARD.MEMBER_ID = SCORE.MEMBER_ID) AS CMS
ON RCTO.CARD_ID = CMS.CARD_ID
WHERE RCTO.RANK = 1;
```

2. Verify count in lookup_data_hbase table.

select count(*) from lookup_data_hbase;

3. Verify some data in lookup_data_hbase table.

select * from lookup_data_hbase limit 10;

4. Start HBase shell from command prompt. In HBase, check count in lookup_data_hive table.

count 'lookup_data_hive';

5. In HBase, check data in lookup_data_hive table.

scan 'lookup_data_hive'