****PLEASE NOTE: Task 1 to 4 remain same as submitted in Logic.pdf for Mid Submission. There is 1 minor change made on page 4. Please check. Hence, adding all the Tasks 1 -7.

Task 1: Load the transactions history data (card_transactions.csv) in a NoSQL database and create a look-up table with columns specified earlier in the problem statement in it.

Pre-step to create the directories which will host the data in HDFS:

From "root" user , switch to HDFS user: **su – hdfs**

Command to create directory in HDFS: **hdfs dfs -mkdir /capstone**

Command to create sub-directory under capstone: **hdfs dfs -mkdir /capstone/data**

Change the owner of the directory created in above step to root:

hdfs dfs -chown -R root /capstone

Using the "put" command, add the card_transaction.csv file from /home/ec2-user into the HDFS location created in the previous step :

hdfs dfs -put /home/ec2-user/card_transactions.csv /capstone/data/

Switch to "root" user. Launch Hive CLI with command : **hive**

This will be a staging table to get the all the raw data in input csv in HDFS:

```
CREATE TABLE IF NOT EXISTS card_transaction_staging(
card_id string,
member_id string,
amount double,
postcode string,
pos_id string,
transaction_dt string,
status string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("separatorChar" = ",")
tblproperties ("skip.header.line.count"="1");
```

Loading the data in this staging table:

```
load data inpath '/capstone/data/card_transactions.csv' into table
card_transaction_staging;
```

Checking the data:

select * from card_transaction_staging limit 10;

```
hive> select * from card_transaction_staging limit 10;
OK
348702330256514 000037495066290 9084849 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 330148  33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 136052  33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 4310362 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 9097094 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 2291118 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 4900011 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 633447  33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 6259303 33946   614677375609919 11-02-2018 00:00
:00     GENUINE
348702330256514 000037495066290 369067  33946   614677375609919 11-02-2018 00:00
:00     GENUINE
Time taken: 0.069 seconds, Fetched: 10 row(s)
```

select count(1) from card_transaction_staging;

53292

```
hive> select count(1) from card_transaction_hive;
Query ID = root_20181201130202_d2cd48b9-b630-499b-ac76-4154f6f26ced
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1543668676487_0001, Tracking URL = http://ip-172-31-46-84.ec2.internal:8088/proxy/application_1543668676487_0001/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1543668676487_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-12-01 13:02:49,836 Stage-1 map = 0%,  reduce = 0%
2018-12-01 13:02:56,186 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.02 sec
2018-12-01 13:03:03,492 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.26 sec
MapReduce Total cumulative CPU time: 6 seconds 260 msec
Ended Job = job_1543668676487_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.26 sec   HDFS Read: 4837633 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 260 msec
OK
53292
```

Creating an ORC table for better efficiency which will be accessed in all subsequent transactions:

```
CREATE EXTERNAL TABLE IF NOT EXISTS card_transaction_ext(
`card_id` string,
`member_id` string,
`amount` double,
`postcode` string,
`pos_id` string,
`transaction_dt` timestamp,
`status` string)
STORED AS ORC
LOCATION 's3a://capstone-deepika/tables'
tblproperties ("orc.compress"="SNAPPY");
```

```
hive> desc card_transactions_ext;
OK
card_id                 string
member_id               string
amount                  double
postcode                string
pos_id                  string
transaction_dt          timestamp
status                  string
Time taken: 0.098 seconds, Fetched: 7 row(s)
```

Loading the data in ORC table from staging table, ensuring the timestamps are in appropriate format:

From card_transaction_staging
insert overwrite table card_transaction_ext
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;

```
hive> From card_transaction_hive
    > insert overwrite table card_transactions_ext
    > 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;
Query ID = root_20181201131010_9d5a3493-6752-4705-a51f-e08e65d6b725
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1543668676487_0002, Tracking URL = http://ip-172-31-46-84.ec2.internal:8088/proxy/application_1543668676487_0002/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1543668676487_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2018-12-01 13:10:28,008 Stage-1 map = 0%,  reduce = 0%
2018-12-01 13:10:38,445 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 13.78 sec
MapReduce Total cumulative CPU time: 13 seconds 780 msec
Ended Job = job_1543668676487_0002
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Loading data to table default.card_transactions_ext
Table default.card_transactions_ext stats: [numFiles=1, numRows=53292, totalSize=1018766, rawDataSize=27925008]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 13.78 sec   HDFS Read: 4835040 HDFS Write: 94 SUCCESS
Total MapReduce CPU Time Spent: 13 seconds 780 msec
OK
Time taken: 28.282 seconds
```

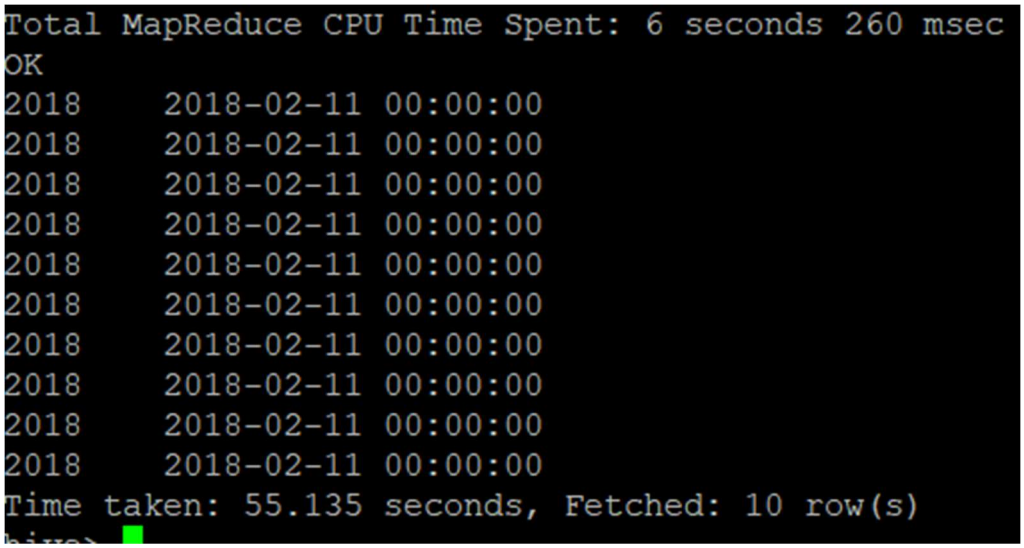
Checking the loaded data:

select * from card_transaction_ext limit 10;

```
hive> select * from card_transaction_ext limit 10;
OK
348702330256514 000037495066290 9084849.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 330148.0        33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 136052.0        33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 4310362.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 9097094.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 2291118.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 4900011.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 633447.0        33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 6259303.0       33946   614677375609919 2018-02-11 00:00:00     G
ENUINE
348702330256514 000037495066290 369067.0        33946   614677375609919 2018-02-11 00:00:00     G
```

Checking the datetime format of the timestamp:

select year(transaction_dt), transaction_dt from card_transaction_ext limit 10;



NOTE: I would be using HBase as the NoSQL database for this problem.

Creating the Hive-Hbase integrated mapping table which will have all the card_transactions data. This table will have a composite row-key -> [combination of member_id , amount , transaction_dt] for uniquely identifying each row :

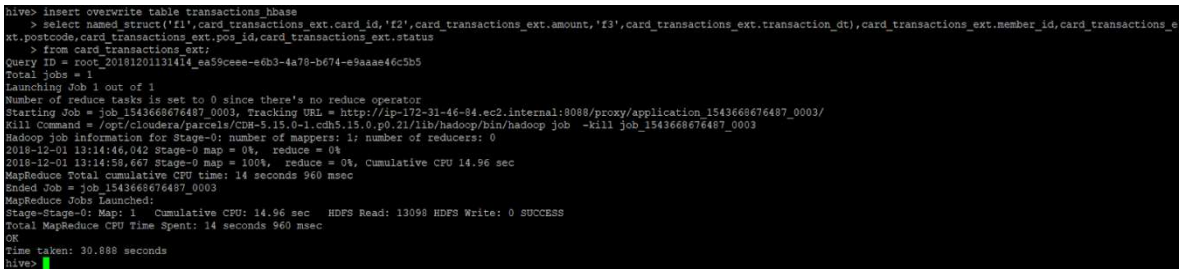
******PLEASE NOTE – As part of final submission, I am making a small change to below table creation script. The change is marked in PURPLE . This has been done to specify the delimiter for the composite key I have created with member_id + amount + transaction_dt. Without this delimiter, HBASE is using the default delimiter , the records being entered were not properly stored in Hbase table (transactions_hive) :**

Reference:
https://learn.upgrad.com/v/course/78/question/101970/answer/436982/comment/105917

```
Create table transactions_hbase(key struct<`member_id`:string, `amount`:double,
`transaction_dt`:timestamp>,`card_id` string,`postcode` string,`pos_id` string,`status`
string)
ROW FORMAT DELIMITED
COLLECTION ITEMS TERMINATED BY '_'
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES
("hbase.columns.mapping"=":key,ctfamily:card_id,ctfamily:postcode,ctfamily:pos_id,ctfa
mily:status")
TBLPROPERTIES ("hbase.table.name"="transactions_hive");
```

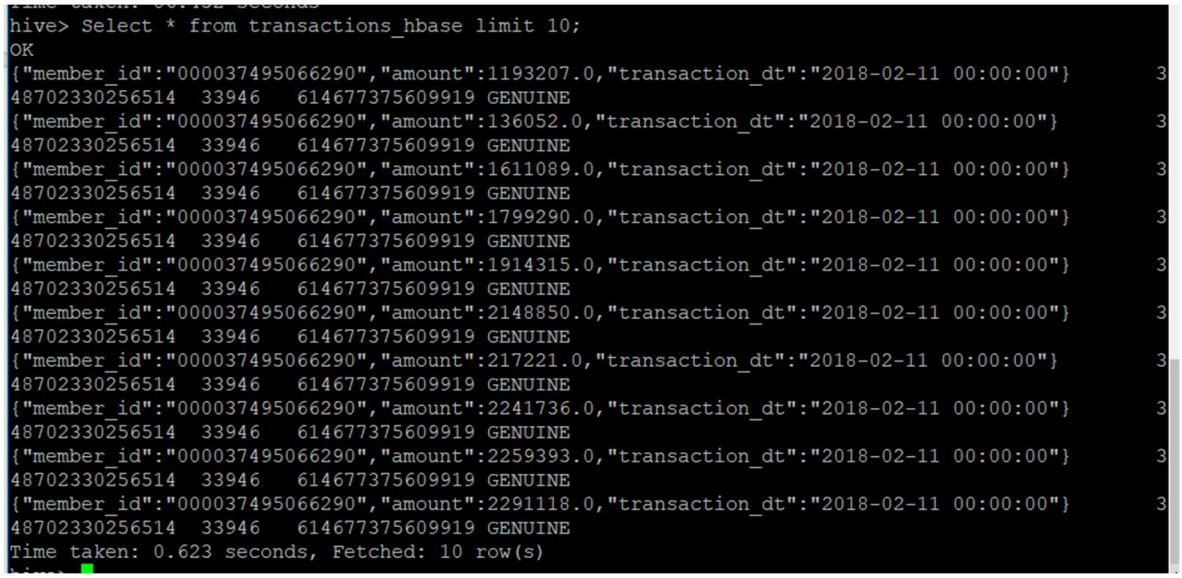
Populating the Hbase and Hive table, directly from Hive:

```
insert overwrite table transactions_hbase
select
named_struct('member_id',card_transaction_ext.member_id,'amount',card_transaction_ext.amount,'transaction_dt',card_transaction_ext.transaction_dt),card_transaction_ext.card_id,card_transaction_ext.postcode,card_transaction_ext.pos_id,card_transaction_ext.status
from card_transaction_ext;
```



Checking the populated data:

```
Select * from transactions_hbase limit 10;
```



Creating a look-up table which will again have Hbase and Hive integration. This table has columns as illustrated in the problem statement – card_id , UCL , score , postcode , transaction_dt :

```
create table lookup_hbase(`card_id` string,`ucl` double,`score` bigint,`postcode` string,
`transaction_dt` timestamp)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,ctfamily:ucl,ctfamily:score,ctfamily:postcode,ctfamily:transaction_dt")
TBLPROPERTIES ("hbase.table.name" = "lookup_hive");
```

Task 2: Write a script to ingest the relevant data from AWS RDS to Hadoop

To run below commands from root user: **hdfs dfs -chown -R root:hive /capstone**

To load data from member_score table stored in RDS into HDFS, use below Sqoop command:

sqoop import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.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/member_score'

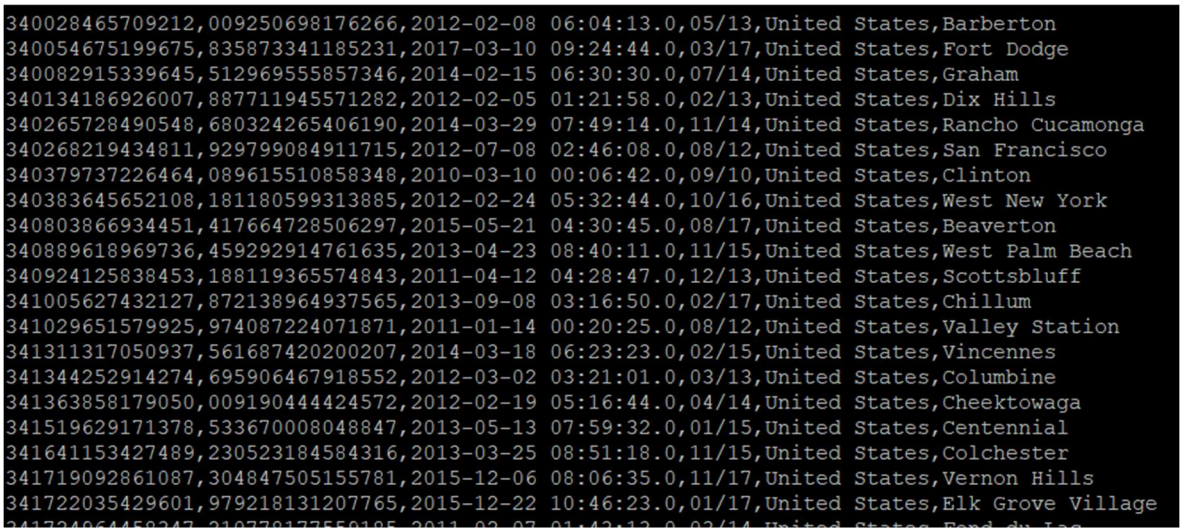
```
[root@ip-172-31-46-84 ~]# hdfs dfs -cat /capstone/member_score/*
000037495066290,339
000117826301530,289
001147922084344,393
001314074991813,225
001739553947511,642
003761426295463,413
004494068832701,217
006836124210484,504
006991872634058,697
007955566230397,372
008732267588672,213
008765307152821,399
009136568025042,308
009190444424572,559
009250698176266,233
009873334520465,298
011716573646690,249
011877954983420,497
012390918683920,407
012731668664932,612
012871894714576,689
015235737327750,244
```

To load data from card_member table stored in RDS into HDFS, use below Sqoop command:

sqoop import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table card_member --null-string 'NA' --null-non-string '\\N' --delete-target-dir --target-dir /capstone/card_member

Checking the data:

hdfs dfs -cat /capstone/card_member/*



```
340028465709212,009250698176266,2012-02-08 06:04:13.0,05/13,United States,Barberton
340054675199675,835873341185231,2017-03-10 09:24:44.0,03/17,United States,Fort Dodge
340082915339645,512969555857346,2014-02-15 06:30:30.0,07/14,United States,Graham
340134186926007,887711945571282,2012-02-05 01:21:58.0,02/13,United States,Dix Hills
340265728490548,680324265406190,2014-03-29 07:49:14.0,11/14,United States,Rancho Cucamonga
340268219434811,929799084911715,2012-07-08 02:46:08.0,08/12,United States,San Francisco
340379737226464,089615510858348,2010-03-10 00:06:42.0,09/10,United States,Clinton
340383645652108,181180599313885,2012-02-24 05:32:44.0,10/16,United States,West New York
340803866934451,417664728506297,2015-05-21 04:30:45.0,08/17,United States,Beaverton
340889618969736,459292914761635,2013-04-23 08:40:11.0,11/15,United States,West Palm Beach
340924125838453,188119365574843,2011-04-12 04:28:47.0,12/13,United States,Scottsbluff
341005627432127,872138964937565,2013-09-08 03:16:50.0,02/17,United States,Chillum
341029651579925,974087224071871,2011-01-14 00:20:25.0,08/12,United States,Valley Station
341311317050937,561687420200207,2014-03-18 06:23:23.0,02/15,United States,Vincennes
341344252914274,695906467918552,2012-03-02 03:21:01.0,03/13,United States,Columbine
341363858179050,009190444424572,2012-02-19 05:16:44.0,04/14,United States,Cheektowaga
341519629171378,533670008048847,2013-05-13 07:59:32.0,01/15,United States,Centennial
341641153427489,230523184584316,2013-03-25 08:51:18.0,11/15,United States,Colchester
341719092861087,304847505155781,2015-12-06 08:06:35.0,11/17,United States,Vernon Hills
341722035429601,979218131207765,2015-12-22 10:46:23.0,01/17,United States,Elk Grove Village
```

Creating staging table in Hive for member score data:

```
CREATE TABLE IF NOT EXISTS member_score(
member_id string,
score bigint
)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH
SERDEPROPERTIES ("separatorChar" = ",");
```

Loading data into staging table:

```
load data inpath '/capstone/member_score' overwrite into table member_score;
```

Creating staging table in Hive for card member data :

```
CREATE TABLE IF NOT EXISTS card_member(
card_id string,
member_id string,
member_joining_dt timestamp,
card_purchase_dt string,
country string,
city string
)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH
SERDEPROPERTIES ("separatorChar" = ",");
```

Loading data into staging table:

```
load data inpath '/capstone/card_member' overwrite into table card_member;
```

Creating an external ORC table for member_score to have data stored for all subsequent uses:

```
CREATE EXTERNAL TABLE IF NOT EXISTS member_score_ext(
member_id string,
score bigint
)
STORED AS ORC location 's3a://capstone-deepika/member_score'
tblproperties ("orc.compress"="SNAPPY");
```

Loading member_score data in ORC table:

```
From member_score
insert overwrite table member_score_ext
Select member_id,score;
```

```
hive> select * from member_Score_ext limit 10;
OK
000037495066290 339
000117826301530 289
001147922084344 393
001314074991813 225
001739553947511 642
003761426295463 413
004494068832701 217
006836124210484 504
006991872634058 697
007955566230397 372
Time taken: 0.68 seconds, Fetched: 10 row(s)
```

Creating an external ORC table for card_member to have data stored for all subsequent uses:

```
CREATE EXTERNAL TABLE IF NOT EXISTS card_member_ext(
card_id string,
member_id string,
member_joining_dt timestamp,
card_purchase_dt string,
country string,
city string
)
STORED AS ORC location 's3a://capstone-deepika/card_member'
tblproperties ("orc.compress"="SNAPPY");
```

Loading card_member data in ORC table:

```
From card_member
insert overwrite table card_member_ext
select card_id , member_id ,member_joining_dt, card_purchase_dt , country , city;
```

```
hive> select * from card_member_ext limit 10;
OK
340028465709212 009250698176266 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
340134186926007 887711945571282 2012-02-05 01:21:58     02/13   United States   Dix Hills
340265728490548 680324265406190 2014-03-29 07:49:14     11/14   United States   Rancho Cucamonga
340268219434811 929799084911715 2012-07-08 02:46:08     08/12   United States   San Francisco
340379737226464 089615510858348 2010-03-10 00:06:42     09/10   United States   Clinton
340383645652108 181180599313885 2012-02-24 05:32:44     10/16   United States   West New York
340803866934451 417664728506297 2015-05-21 04:30:45     08/17   United States   Beaverton
340889618969736 459292914761635 2013-04-23 08:40:11     11/15   United States   West Palm Beach
Time taken: 0.311 seconds, Fetched: 10 row(s)
hive>
```

Task 3: Write a script to calculate the moving average and standard deviation of the last 10 transactions for each card_id for the data present in Hadoop and NoSQL database. The script should be able to extract and feed the other relevant data ('postcode', 'transaction_dt', 'score', etc.) for the look-up table along with card_id and UCL

Creating 1st table that will provide the last 10 transactions for each card_id :

```
CREATE EXTERNAL TABLE IF NOT EXISTS ranked_table(
card_id string,
amount double,
postcode string,
transaction_dt timestamp,
rank int)
STORED AS ORC location 's3a://capstone-deepika/ranked_table'
tblproperties ("orc.compress"="SNAPPY");
```

Inserting data into this 1st table : *[You can load data from card_transaction_ext / transactions_hbase table at this point because both will have same data after first load]*

```
INSERT INTO ranked_table
select * from
(select * , rank() OVER(PARTITION BY card_id ORDER BY transaction_dt DESC)rank from
(select card_id, amount , postcode , transaction_dt from card_transaction_ext where
status = 'GENUINE')a
```

```
GROUP BY card_id, amount , postcode , transaction_dt)b where rank <= 10;
```

Note: *While wrangling data, keep the stages in place for better backtracking of your transformations (keeping track of steps) as a best practice. Let's suppose that you are calculating* UCL *in* Hive*. You might first need a raw table, then in the next staging table, you might want to derive the moving average and standard deviation and then the final table with* UCL *value. This helps manage the layers of your wrangled data*

Creating 2nd table that will find average and standard deviation of last 10 transactions of every card:

```
CREATE EXTERNAL TABLE IF NOT EXISTS wrangled_staging_tbl(
card_id string,
average double,
standard_deviation double)
STORED AS ORC location 's3a://capstone-deepika/wrangled_staging_tbl'
tblproperties ("orc.compress"="SNAPPY");
```

Inserting data into this table:

```
INSERT INTO wrangled_staging_tbl
Select card_id, AVG(amount), STDDEV(amount) from ranked_table group by card_id;
```

Creating a third table to derive UCL based on average and standard deviation calculated in previous table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ucl_calc_ext(
card_id string,
ucl double)
STORED AS ORC location 's3a://capstone-deepika/ucl_calc_ext'
tblproperties ("orc.compress"="SNAPPY");
```

Inserting data into this table:

```
INSERT INTO ucl_calc_ext
Select card_id, (average + 3* standard_deviation) as UCL from wrangled_staging_tbl;
```

Creating another table to get score for each card_id :

```
create external table if not exists lookup_interim_join(
card_id string,
member_id string,
score bigint)
STORED AS ORC location 's3a://capstone-deepika/lookup_interim_join'
tblproperties ("orc.compress"="SNAPPY");
```

Below query gets the card_id and score derived by joining 2 tables - card_member_ext and member_score_ext. There is no other table which has card_id and score associated, hence joining the 2 available ones to get score for every card_id :

```
INSERT into lookup_interim_join
Select card_member_ext.card_id , card_member_ext.member_id ,
member_score_ext.score
FROM card_member_ext INNER JOIN member_score_ext ON
(card_member_ext.member_id = member_score_ext.member_id);
```

This will be the final table which will have all the columns required for lookup:

```
CREATE EXTERNAL TABLE IF NOT EXISTS lookup_joined_ext(
card_id string,
UCL double,
score bigint,
postcode string,
transaction_dt timestamp
)
STORED AS ORC location 's3a://capstone-deepika/lookup_joined_ext'
tblproperties ("orc.compress"="SNAPPY");
```

Below query gets all the required lookup columns in 1 table - card_id , ucl , score , postcode and transaction_dt. Join is performed on 3 tables - ucl_calc_ext [card_id , ucl] , lookup_interim_join [card_id , member_id , score] and ranked_table [card_id, a mount,postcode , transaction_dt , rank]. card_id is the common column in all these tables :

```
INSERT INTO lookup_joined_ext
select ucl_calc_ext.card_id , ucl_calc_ext.ucl , lookup_interim_join.score ,
ranked_table.postcode , ranked_table.transaction_dt
FROM ucl_calc_ext JOIN lookup_interim_join ON (ucl_calc_ext.card_id =
lookup_interim_join.card_id)
JOIN ranked_table ON (ranked_table.card_id = lookup_interim_join.card_id  AND
ranked_table.rank = 1);
```

Check the data :

```
Select * from lookup_joined_ext limit 100;
```

```
5208315353552491        1.3695877175952177E7    327     17768   2018-01-26 15:33:07
5208579982289235        1.2240401464866431E7    220     75849   2018-01-18 14:26:10
5208913608059252        1.4220609033993615E7    485     56575   2018-01-29 12:01:13
5211331689203399        1.2526285638430517E7    360     98554   2018-01-22 07:58:17
5211789892119498        1.5500915148854515E7    638     45894   2018-01-29 15:31:53
5218975136635638        1.2970031541103944E7    475     42366   2018-01-23 01:31:39
5219257261928206        1.2811402304329943E7    474     36568   2018-01-23 03:17:00
5221159076474119        1.1278856929484006E7    525     43747   2017-01-31 06:39:21
5221181828348995        1.3030012644738147E7    311     18950   2018-01-31 07:54:36
5221229593682054        1.5638090534733517E7    419     58335   2018-01-27 09:06:29
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     71860   2018-01-11 00:00:00
5226069160942563        1.3286741358658955E7    617     22989   2018-01-11 00:00:00
```

The final step is to feed the data into look-up table in HBase. This table was created as final step of Task 1:

insert overwrite table lookup_hbase
select card_id,ucl,score,postcode,transaction_dt from lookup_joined_ext;

```
Hadoop job information for Stage-0: number of mappers: 1; number of reducers: 0
2018-12-01 21:26:34,221 Stage-0 map = 0%,  reduce = 0%
2018-12-01 21:26:56,243 Stage-0 map = 100%,  reduce = 0%, Cumulative CPU 9.88 sec
MapReduce Total cumulative CPU time: 9 seconds 880 msec
Ended Job = job_1543689740360_0025
MapReduce Jobs Launched:
Stage-Stage-0: Map: 1   Cumulative CPU: 9.88 sec   HDFS Read: 12941 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 880 msec
OK
Time taken: 71.879 seconds
hive> select * from lookup_hbase limit 5;
OK
340028465709212 1.5592998236442273E7    233     24658   2018-01-02 03:25:35
340054675199675 1.4544873848898392E7    631     50140   2018-01-15 19:43:23
340082915339645 1.5131197317721393E7    407     17844   2018-01-26 19:03:47
340134186926007 1.5397940821958443E7    614     67576   2018-01-18 23:12:50
340265728490548 1.543114695203365E7     202     72435   2018-01-21 02:07:35
Time taken: 0.445 seconds, Fetched: 5 row(s)
```

Task 4: Set up a job scheduler to schedule the scripts run after every 4 hours. The job should take the data from the NoSQL database and AWS RDS and perform the relevant analyses as per the rules and should feed the data in the look-up table

Please create this table through Hive CLI which will hold the data for the sqoop incremental loads for card_member table:

```
CREATE TABLE IF NOT EXISTS card_member_incremental(
card_id string,
member_id string,
member_joining_dt timestamp,
card_purchase_dt string,
country string,
city string
)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH
SERDEPROPERTIES ("separatorChar" = ",");
```

This above table will initially be populated with all records coming in from card_member table (AWS RDS) through Sqoop incremental load job run. But on all subsequent job runs, it will only get in any new records created / modified[Delta of records] in card_member table.

Hence this table will always hold the most latest replica of AWS RDS table – card_member table as it will be getting incremental updates/inserts every 4 hours when Sqoop incremental job is run.

***NOTE**: I have added the Hive query to receive only incremental records in card_member_ext table[Hive table] after each Sqoop import job run in hiveScript.sql file*

Create these two jobs in Sqoop meta store for regular 4 hourly imports:

```
sqoop job --create inc_upsert_card_member --meta-connect jdbc:hsqldb:hsql://ip-172-31-46-84.ec2.internal:16000/sqoop -- import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table card_member --null-string 'NA' -m 1 --incremental lastmodified --check-column member_joining_dt --last-value 0 --merge-key card_id --target-dir /capstone/card_member_incremental
```

```
sqoop job --create update_member_score --meta-connect jdbc:hsqldb:hsql://ip-172-31-46-84.ec2.internal:16000/sqoop -- import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwmz.us-east-1.rds.amazonaws.com/cred_financials_data --username upgraduser --password upgraduser --table member_score --null-string 'NA' -m 1 --delete-target-dir --target-dir /capstone/member_score
```

Job.properties:

```
nameNode=hdfs://ip-172-31-46-84.ec2.internal:8020
jobTracker=ip-172-31-46-84.ec2.internal:8032
oozie.use.system.libpath=True
wfdir=${nameNode}/user/${user.name}/oozie_sqoop
hivescript=${wfdir}/app/hiveScript.sql
queueName=default

oozie.coord.application.path=${wfdir}/coordinator.xml
start=2018-12-08T15:02Z
end=2019-01-31T01:00Z
workflowpath=${wfdir}/app/workflow.xml
```

coordinator.xml:

```
<coordinator-app name="capstone-coord" start="${start}" end="${end}"
frequency="0 */4 * * *" timezone="UTC" xmlns="uri:oozie:coordinator:0.4">
    <controls>
        <timeout>1</timeout>
        <concurrency>2</concurrency>
        <execution>FIFO</execution>
        <throttle>2</throttle>
    </controls>
    <action>
        <workflow>
            <app-path>${workflowpath}</app-path>
            <configuration>
                <property>
                    <name>jobTracker</name>
                    <value>${jobTracker}</value>
                </property>
                <property>
                    <name>nameNode</name>
                    <value>${nameNode}</value>
                </property>
                <property>
                    <name>queueName</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
        </workflow>
    </action>
</coordinator-app>
```

Workflow.xml:

```
<workflow-app name="capstone" xmlns="uri:oozie:workflow:0.4">
<start to="card_member_import"/>
<action name="card_member_import">
      <sqoop xmlns="uri:oozie:sqoop-action:0.2">
        <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <job-xml>sqoop-site.xml</job-xml>
      <configuration>
           <property>
                <name>fs.hdfs.impl.disable.cache</name>
                <value>true</value>
           </property>
           <property>
                <name>mapred.job.queue.name</name>
                <value>default</value>
           </property>
      </configuration>
      <command>job --exec inc_upsert_card_member --meta-connect jdbc:hsqldb:hsql://ip-
172-31-46-84.ec2.internal:16000/sqoop</command>
     </sqoop>
      <ok to="member_score_import"/>
      <error to="kill"/>
</action>


<action name="member_score_import">
      <sqoop xmlns="uri:oozie:sqoop-action:0.2">
        <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <job-xml>sqoop-site.xml</job-xml>
      <configuration>
           <property>
                <name>fs.hdfs.impl.disable.cache</name>
                <value>true</value>
           </property>
           <property>
                <name>mapred.job.queue.name</name>
                <value>default</value>
           </property>
      </configuration>
      <command>job --exec update_member_score --meta-connect jdbc:hsqldb:hsql://ip-
172-31-46-84.ec2.internal:16000/sqoop</command>
     </sqoop>
      <ok to="load_hbase_lookup"/>
      <error to="kill"/>
</action>
```

```
<action name="load_hbase_lookup">
      <hive2 xmlns="uri:oozie:hive2-action:0.1">
         <job-tracker>${jobTracker}</job-tracker>
         <name-node>${nameNode}</name-node>
         <jdbc-url>jdbc:hive2://ip-172-31-46-84.ec2.internal:10000/default</jdbc-url>
         <script>${hivescript}</script>
      </hive2>
      <ok to="finish"/>
      <error to="kill"/>
   </action>

   <kill name="kill">
      <message>Your job failed!</message>
   </kill>
<end name="finish"/>
</workflow-app>
```

hiveScript.sql :

```
load data inpath '/capstone/card_member_incremental' overwrite into table
card_member_incremental;

load data inpath '/capstone/member_score' overwrite into table member_score;

insert into card_member_ext
select card_member_incremental.* from card_member_incremental LEFT JOIN
card_member_ext ON (card_member_incremental.card_id = card_member_ext.card_id)
WHERE card_member_ext.card_id IS NULL;

insert overwrite table member_score_ext
select * from member_score;

INSERT overwrite table ranked_table select * from (select * , rank() OVER(PARTITION BY
card_id ORDER BY transaction_dt DESC)rank from (select card_id, key.amount as amount ,
postcode , key.transaction_dt as transaction_dt from transactions_hbase where status =
'GENUINE')a
GROUP BY card_id, amount , postcode , transaction_dt)b where rank <= 10;

INSERT overwrite table wrangled_staging_tbl
Select card_id, AVG(amount), STDDEV(amount) from ranked_table group by card_id;

INSERT overwrite table ucl_calc_ext
Select card_id, (average + 3* standard_deviation) as UCL from wrangled_staging_tbl;



INSERT overwrite table lookup_interim_join
```

```
Select card_member_ext.card_id , card_member_ext.member_id ,
member_score_ext.score
FROM card_member_ext INNER JOIN member_score_ext ON
(card_member_ext.member_id = member_score_ext.member_id);


INSERT overwrite table lookup_joined_ext
select ucl_calc_ext.card_id , ucl_calc_ext.ucl , lookup_interim_join.score ,
ranked_table.postcode , ranked_table.transaction_dt
FROM ucl_calc_ext JOIN lookup_interim_join ON (ucl_calc_ext.card_id =
lookup_interim_join.card_id)
JOIN ranked_table ON (ranked_table.card_id = lookup_interim_join.card_id  AND
ranked_table.rank = 1);


insert overwrite table lookup_hbase
select card_id,ucl,score,postcode,transaction_dt from lookup_joined_ext;
```

NOTE: *Please follow each step mentioned in IMP-OoziePreRequisitesAndExecutionGuide.pdf to execute the Oozie workflow successfully.*

Task 5: Create a streaming data processing framework which ingests real-time POS transaction data from Kafka. The transaction data is then validated based on the three rules' parameters (stored in the NoSQL database) discussed above.

Please refer FraudAnalysis.zip

Different Layers in the framework are:

- ***Driver Class***
- ***Business Logic Class***
- ***Data Access Object Class***
- ***Utility Class***

The driver class is named as – KafkaStreamingIngestion.

This has the main method which calls JavaStreamingContext. This in turn sets up the context with SparkConf and batch duration for the incoming D-Streams (internally RDD's).
The application subscribes to Kafka stream as consumer for topic - `transactions-topic-verified.` The stream being generated by Kafka is then held by a Collection<String> of topics.

It is then converted into JavaDStreams which are further processed to solve our problem statement.
For the DStreams having Kafka topics, we are interested only in Value part. ***Transformation Used: Map***
From the value part also, we are interested in only those records which begin with `{\"card_id\":` as there are many more records in the steam which are just 0's. ***Transformation Used: Filter***
Once the record is obtained, the constructor of FraudDetection class is invoked. ***Transformation Used: Map***
After this transformation, each record in the JavaDStream becomes an object of type FraudDetection and then invokes the method in FraudDetection class for updating NoSQL DB with the new record.

Business Logic class is named as – FraudDetection.

The parameterized constructor of FraudDetection class takes the DStreams from Driver class as JSONobject and parses the object and then initializes the class variables with the incoming values.
Methods in DAO class are like: ➢ getSpeed ➢ validateTransaction ➢ updateHbaseTable
The updateHBaseTable method of FraudDetection class uses the member variables , initialized by constructor above with incoming transaction values, and validates the transaction record based on 3 parameters for marking it as GENUINE or FRAUD before inserting it in NoSQL DB table.
The validateTransaction method in FraudDetection class uses below Validation Rules for verifying the authenticity of incoming transactions: ➢ If the member score is less than 200, the transaction needs to be rejected. ➢ The transaction amount of current transaction cannot exceed beyond UCL set for this card based on last 10 genuine transactions. ➢ If the speed between 2 transactions is more than 1km/4sec = 0.25km/second, it is Fraud.
Any transaction which comes clean from each one of these conditions, is deemed – GENUINE else it is FRAUD
NOTE: It was found that many incoming records from Kafka stream contained older transaction dates than what is present in lookup table in NoSQL DB. This was skewing the data to have more FRAUD transactions. Hence, I have used Math.abs() method to get the absolute difference between 2 dates so as to achieve correct results for Speed and hence correct classification.
The entire information about the transaction is then sent to DAO Class as FraudDetection object.

Data Access Object (DAO) class is named as – NoSQLDBConAndUpdates.

By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the NoSQL database (Hbase)
Methods in DAO class are like: ➢ getScore ➢ getUCL ➢ getPostCode ➢ getTransactionDt ➢ updateTable ➢ updateLookUpTable
This class helps the FraudDetection class for getting values of ucl , score , postcode and transaction date from lookup_hive table which is a quick reference table in NoSql DB. These values help the Business logic class's methods validate a transaction.
This class also helps in inserting new records and updating existing ones in the below tables based on the incoming transaction records processed by Business Logic class Transactions_hive Lookup_hive These two are tables in HBase which have a mapping created in Hive.
Each object received by DAO class is first checked for the data it contains. If any nulls are present in the primary key of the table where inserts are being made (transactions_hive), the record will not get processed. Only if all primary key components are present, a new record will be inserted with correct status as – GENUINE / FRAUD
If the status of the record is GENUINE, the other table (lookup_hive) will then be updated with current transaction's postcode and transaction_date.

Utility class is named as – DistanceUtility

This is a Utility class that reads file zipCodePosId.csv and using the same if two zipcodes are provided, it returns distances. The getSpeed method in Business Logic layer uses this method to calculate the distance between 2 postcodes – the one in current transaction record and the other one retrieved from the lookup table which had the latest postcode for a card transaction This is a key parameter for validating a transaction as GENUINE / FRAUD.
Methods in DistanceUtility class are like: ➢ getDistanceViaZipCode

Task 6: Update the transactions data along with the status (Fraud/Genuine) in the card_transactions table.

Method **updateTable(FraudDetection transactionData)** in DAO class (***NoSQLDBConAndUpdates***) updates the HBASE table named – "**transactions_hive**" with new records as follows:

******Please note: Kindly take care of specifying the delimiter when defining a composite key during HBASE-HIVE mapped table creation. Without that, HBase table takes its default delimiter and does not store the records correctly. I have specified the change, I have made after mid-submission, above in page 4.**

```
public static void updateTable(FraudDetection transactionData) throws IOException {

    try {
        if( "".equals(transactionData.getMember_id()) || transactionData.getMember_id().equals(null) ||
                Double.valueOf(transactionData.getAmount()).equals(null) || transactionData.getTransaction_dt() == null ) {

            System.out.println("One of the values in composite key - member_id  or amount or transaction_dt is blank or null. "
                    + "Moving to next record..");

        }else {

            //Call the getHbaseAdmin() method only if connection is not already established
            if(hBaseAdmin == null) {
                hBaseAdmin = getHbaseAdmin();
            }

            //Converting the incoming object date to specified format
            DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String val = dateFormat.format(transactionData.getTransaction_dt());

            //System.out.println("Connecting HBase now to update the transaction record in transactions table.....");
            String tblName1 = "transactions_hive";

            // Check if table exists
            if (hBaseAdmin.tableExists(TableName.valueOf(tblName1))) {
                Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(tblName1));
```

```
            /********** adding a new row **********/

            /*
             * Since I have created a composite row key for transactions_hive table,
             * the key is created as - member_id,amount,transaction_dt
             *
             */

            Put p = new Put(Bytes.toBytes(transactionData.getMember_id() + "_" +
                    transactionData.getAmount() + "_" + val));


            //Adding other column values in each column
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("card_id"), Bytes.toBytes(transactionData.getCard_id()));
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("postcode"), Bytes.toBytes(transactionData.getPostcode()));
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("pos_id"), Bytes.toBytes(transactionData.getPos_id()));
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("status"), Bytes.toBytes(transactionData.getStatus()));
            htable.put(p);
            System.out.println("Hbase Table '" + tblName1 + "' is Populated");
        }
        else {
            System.out.println("The HBase Table named '" + tblName1 + "' doesn't exists.");
        }
        //Checking if status was Genuine, send it for updating the lookup_hive table
        if(transactionData.getStatus().equals("GENUINE")) {
            System.out.println("Since the transaction is GENUINE, hence updating the lookup table with current details...");
            updateLookUpTable(transactionData.getCard_id() ,
                    transactionData.getPostcode() , transactionData.getTransaction_dt());
        }

    }
}catch(Exception e) {
    e.printStackTrace();
}
}
```

Task 7: Store the 'postcode' and 'transaction_dt' of the current transaction in the look-up table in the NoSQL database.

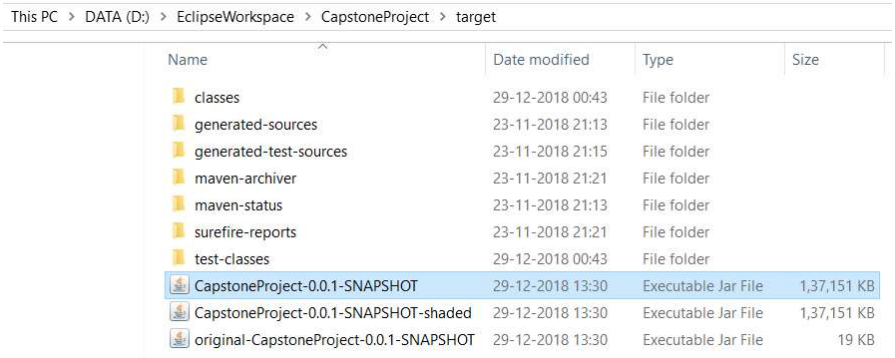
- Method ***updateLookupTable(card_id , postcode , transactionDate)*** in DAO class (***NoSQLDBConAndUpdates***) updates the HBASE table named – "***lookup_hive***" as follows:

- For the card_id in the incoming transaction record, the postcode and transaction_dt of the existing record is updated only if the transaction status was validated as GENUINE by FraudDetection class. Card_Id is primary key of lookup_hive table.

```
private static void updateLookUpTable(String card_id , String latestPostCode , Date latestTransactionDt) throws IOException {

    try {

        //Converting the incoming object date to specified format
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String val = dateFormat.format(latestTransactionDt);

        //Call the getHbaseAdmin() method only if connection is not already established
        if(hBaseAdmin == null) {
            hBaseAdmin = getHbaseAdmin();
        }

        //System.out.println("Connecting HBase now for updating lookup_hive table with latest post code and transaction date...");
        String table = "lookup_hive";


        // Check if table exists
        if (hBaseAdmin.tableExists(TableName.valueOf(table))) {
            Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(table));

            //Instantiating the Put class to row name. "card_id" is the row key in lookup_hive table
            Put p = new Put(Bytes.toBytes(card_id));

            //Updating the cell values
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("postcode"), Bytes.toBytes(latestPostCode));
            p.addColumn(Bytes.toBytes("ctfamily"), Bytes.toBytes("transaction_dt"), Bytes.toBytes(val));

            // Saving the put Instance to the HTable.
            htable.put(p);
            System.out.println("Postcode and TransactionDt updated for card id :  '"+ card_id + "' in lookup_hive HBase table");

        }else {
            System.out.println("The HBase Table named '" + table + "' doesn't exists.");
        }
    }catch(Exception e) {
        e.printStackTrace();
    }

}
```

Execution Steps to run the framework:

1. Create a fat jar of the Maven solution by including Maven Shade Plugin in the POM.xml file. Build the project : Maven -> Update Project and then run: Run As -> Maven Install
2. The fat jar gets created under Target directory in the project folder (Project folder in Eclipse Workspace)



3. Use WinSCP to copy this jar to AWS EC2 instance. Put it under /root directory.
4. Also copy the zipCodePosId.csv file to AWS EC2 instance. Put this also in /root.
5. Launch Putty terminal as ec2-user, change to root user -> **sudo -i**
6. On another Putty instance, become the root user and go to hbase shell with command **hbase shell**

7. On hbase shell, execute the below command:
   count 'transactions_hive', INTERVAL => 10, CACHE => 1000

   The count will be: **53292**
8. On first instance of Putty, execute the following commands in order:
   Please note for spark-2 submit command:

   - Replace the Public DNS IPv4 with your EC2 instance public IPv4. For e.g. **ec2-100-24-28-146.compute-1.amazonaws.com**
   - Replace the file-location with exact location where you have put zipCodePosId.csv For e.g. - **/root/zipCodePosId.csv**
   - You may replace the output file location as desired. I have used: **/tmp/CapstoneProjectOutput.txt**

   export SPARK_KAFKA_VERSION=0.10

   spark2-submit --class com.upgrad.CapstoneProject.KafkaStreamingIngestion --master yarn --deploy-mode client --name FraudDetection --conf "spark.app.id=FraudDetection spark.driver.memory=12g spark.executor.memory=12g spark.executor.instances=2" /root/CapstoneProject-0.0.1-SNAPSHOT.jar <Public DNS IPv4> <file-location> &> /tmp/CapstoneProjectOutput.txt

9. Periodically check on hbase sell for the count, usually in 15-20 min, the count will become: **59367**

```
Current count: 59180, row: 992552823055811_8509100.0_2017-10-10 18:02:40
Current count: 59190, row: 992552823055811_9368443.0_2017-10-07 04:01:12
Current count: 59200, row: 994087875645 92_4194716.0_2018-02-03 06:40:05
Current count: 59210, row: 994983851226493_3515213.0_2018-01-18 03:54:09
Current count: 59220, row: 994983851226493_8635558.0_2017-09-12 21:09:42
Current count: 59230, row: 996411635289270_1310256.0_2016-06-17 21:19:11
Current count: 59240, row: 996411635289270_296301.0_2016-02-03 18:43:46
Current count: 59250, row: 996411635289270_5381248.0_2017-01-12 08:46:29
Current count: 59260, row: 996411635289270_6698340.0_2017-08-07 02:05:44
Current count: 59270, row: 997128952368160_1036903.0_2018-01-29 12:01:13
Current count: 59280, row: 997128952368160_1630620.0_2018-01-29 08:30:33
Current count: 59290, row: 997128952368160_2586660.0_2017-11-11 00:00:00
Current count: 59300, row: 997128952368160_3526280.0_2017-11-11 00:00:00
Current count: 59310, row: 997128952368160_4760248.0_2017-11-17 01:23:58
Current count: 59320, row: 997128952368160_5596131.0_2017-11-15 01:59:54
Current count: 59330, row: 997128952368160_6790715.0_2017-11-02 20:48:49
Current count: 59340, row: 997128952368160_7309585.0_2017-11-16 18:22:38
Current count: 59350, row: 997128952368160_8243967.0_2017-11-11 00:00:00
Current count: 59360, row: 997128952368160_9215394.0_2018-10-11 10:37:48
59367 row(s) in 3.5030 seconds

=> 59367
hbase(main):014:0>
```

10. The output is present in /tmp folder as - **CapstoneProjectOutput.txt**
    Use WinSCP to get it on your local machine. You can convert it into pdf – Output.pdf
11. Now again as root user on EC2 instance, execute this command:

    hive -e 'select * from transactions_hbase;' | sed 's/[\t]/,/g' > /tmp/card_transactions.csv

```
-bash-4.2# whoami
root
-bash-4.2# hive -e 'select * from transactions_hbase;' | sed 's/[\t]/,/g' > /tmp/Deepika_card_transactions.csv
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: Using incremental CMS is deprecated and will likely be removed in a future release
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was removed in 8.0

Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/jars/hive-common-1.1.0-cdh5.15.0.jar!/hive-log4j.properties
OK
Time taken: 6.904 seconds, Fetched: 59367 row(s)
-bash-4.2#
```

12. Use WinSCP to get this card_transactions.csv from /tmp to your local machine.
13. Close the Putty sessions.
14. Stop the EC2 instance.
15. The card_transactions.csv should now be checked for inserted records. On checking, I have received:

    GENUINE : 59201
    FRAUD : 166