

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_staging;
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_staging
> 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;
```

```
Total MapReduce CPU Time Spent: 6 seconds 260 msec
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: 55.135 seconds, Fetched: 10 row(s)
```

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 :

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

```
hive> describe transactions_hbase;
OK
key          struct<member_id:string,amount:double,transaction_dt:timestamp> from deserializer
card_id      string          from deserializer
postcode     string          from deserializer
pos_id       string          from deserializer
status       string          from deserializer
Time taken: 0.21 seconds, Fetched: 5 row(s)
```

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;

```
hive> insert overwrite table transactions_hbase
> select named_struct('f1',card_transactions_ext.card_id,'f2',card_transactions_ext.amount,'f3',card_transactions_ext.transaction_dt),card_transactions_ext.member_id,card_transactions_ext.postcode,card_transactions_ext.pos_id,card_transactions_ext.status
> from card_transactions_ext;
Query ID = root_20181201131414_ea59ceee-e6b3-4a78-b674-e9aaae46c5b5
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_0003, Tracking URL = http://ip-172-31-46-84.ec2.internal:8088/proxy/application_1543668676487_0003/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job -kill job_1543668676487_0003
Hadoop job information for Stage-0: number of mappers: 1; number of reducers: 0
2018-12-01 13:14:46,042 Stage-0 map = 0%, reduce = 0%
2018-12-01 13:14:58,667 Stage-0 map = 100%, reduce = 0%, Cumulative CPU 14.96 sec
MapReduce Total cumulative CPU time: 14 seconds 960 msec
Ended Job = job_1543668676487_0003
MapReduce Jobs Launched:
Stage-Stage-0: Map: 1 Cumulative CPU: 14.96 sec HDFS Read: 13098 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 960 msec
OK
Time taken: 30.888 seconds
hive>
```

Checking the populated data:

Select * from transactions_hbase limit 10;

```
hive> Select * from transactions_hbase limit 10;
OK
{"member_id":"000037495066290","amount":1193207.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":136052.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":1611089.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":1799290.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":1914315.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":2148850.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":217221.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":2241736.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":2259393.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
{"member_id":"000037495066290","amount":2291118.0,"transaction_dt":"2018-02-11 00:00:00"} 3
48702330256514 33946 614677375609919 GENUINE
Time taken: 0.623 seconds, Fetched: 10 row(s)
```

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.cpclxrkdvwzmz.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
015508173543086,176
```

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

```
sqoop import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwzmz.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
341724064458347,210778177550185,2011-02-07 01:42:12.0,02/14,United States,Ford, Ky, Usa

```

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;
```


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:hsq://ip-172-31-46-84.ec2.internal:16000/sqoop -- import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwzmz.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:hsq://ip-172-31-46-84.ec2.internal:16000/sqoop -- import --connect jdbc:mysql://upgradawsrds.cpclxrkdvwzmz.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
wkdir=${nameNode}/user/${user.name}/oozie_sqoop
hivescript=${wkdir}/app/hiveScript.sql
queueName=default

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