Final-Submission – Logic Explanation

NOTES:

1. I have submitted below artifacts in final submission for capstone project:

    1. FraudAnalysis.zip
    2. LogicFinal.pdf
    3. Output.pdf
    4. Output_without_details_of_records.pdf
    5. Output.txt
    6. card_transactions.csv

    I have deliberately provided **Output_without_details_of_records.pdf**. I have displayed all 6075 records received from Kafka topic in the Output.pdf file. Just in case, evaluator directly wants to see logs for processing of transactions then can open Output_without_details_of_records.pdf or evaluator will have to scroll down up to page number 290 to start with the logs for processing of transactions.

    I have deliberately provided **Output.txt** as well. This is the original file which captured the output of spark program. Since line length of a text file is much higher than of a word or pdf file, if you open this Output.txt file in Notepad++ then you will see output with more clarity.

    Please zoom to 180% or 200% to see screenshots with better clarity.

2. As per submission guidelines for final submission point 8, we are supposed to submit screenshots of console output. Since program was running very fast, I was not able to take screenshots. I was able to redirect console output to a file. It has been confirmed by TA in this discussion forum link that we can submit that output file and it will suffice: https://learn.upgrad.com/v/course/119/question/129993

3. This link mentioned in resources section of capstone project, was used for Kafka configuration: https://learn.upgrad.com/v/course/119/question/123157

4. This link mentioned in resources section of capstone project, was used to redirect spark output to a file: https://learn.upgrad.com/v/course/119/question/123151

5. This link mentioned in resources section of capstone project, was used to get rid of issue faced and added export SPARK_KAFKA_VERSION=0.10 : https://learn.upgrad.com/v/course/119/question/123152

6. This link mentioned in resources section of capstone project, was used for Spark-HBase configuration and operations: https://learn.upgrad.com/v/course/119/question/123146

7. Spark and hive commands are executed via ec2-user.

8. DistanceUtility.java provided was used as is as such with addition of following statements:

    At the top of the file: **package com.upgrad.creditcardfrauddetection;**

    In DistanceUtilit() method, at the end: **br.close();**

9. Copy FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar and zipCodePosId.csv in /home/ec2-user directory of ec2 instance and run below commands to execute the program:

   export SPARK_KAFKA_VERSION=0.10

   spark2-submit \
   --class com.upgrad.creditcardfrauddetection.KafkaConsumer \
   --master yarn --deploy-mode client \
   --name CreditCardFraudDetection \
   --conf "spark.app.id=CreditCardFraudDetection spark.driver.memory=12g spark.executor.memory=12g spark.executor.instances=2" \
   /home/ec2-user/FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
   ec2-54-237-193-160.compute-1.amazonaws.com \
   &> /tmp/Output.txt

   NOTE: Highlighted portion is public DNS for my ec2 instance. Passing it as argument rather than hard coding in the program because it keeps on changing with restart of ec2 instance every time.

10. Run below command to extract card_transactions_hbase table into a file:

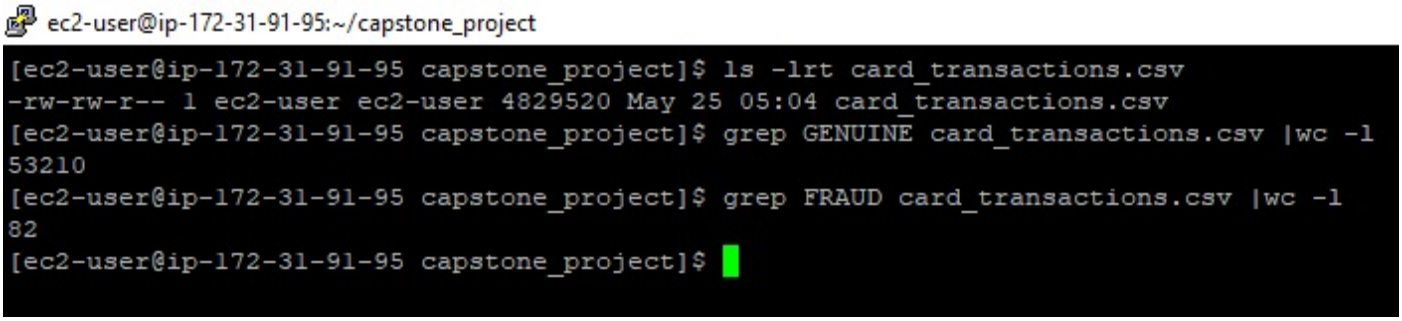
    hive -S -e 'use capstone_project; set hive.cli.print.header=true; select * from card_transactions_hbase' | sed 's/[\t]/,/g' | sed 's/card_transactions_hbase\.//g' | sed '/^WARN:/d' > /tmp/card_transactions.csv

11. Total records in submitted card_transactions.csv are: 59367 (excluding header)

    GENUINE: 59150
    FRAUD: 217

12. Original card_transactions.csv had 53292 records:

    GENUINE: 53210
    FRAUD: 82



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.

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

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

========== Logic Explanation for Task 5 - 6 -7 – Start ==========

FraudAnalysis is the maven project containing 3 java files namely:

1. **DistanceUtility.java**
2. **KafkaConsumer.java**
3. **CreditCardFraudDetection.java**

1. **DistanceUtility.java**

It was already provided and used as is. Following statements were added in it in order to compile:

At the top of the file:

package com.upgrad.creditcardfrauddetection;

In DistanceUtility class, at the end:

br.close();

This is a Utility class that reads file zipCodePosId.csv and using the same if two post codes are provided as input, it returns distance between two post codes.

The getSpeed method in CreditCardFraudDetection class uses this utility to calculate the distance between 2 post codes. One post code in current transaction record and the other one retrieved from the lookup table which had the latest postcode for a card from last GENUINE transaction. This is one of validation rule for validating a transaction as GENUINE / FRAUD.

Below are the screenshots for DistanceUtility class:

```java
package com.upgrad.creditcardfrauddetection;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;

/**
 * Utility class that reads file zipCodePosId.csv and using same if two zip
 * codes are provided, it returns distances.
 *
 */

class ZipCodeData {
    double lat;
    double lon;
    String city;
    String state_name;
    String postId;

    public ZipCodeData(double lat, double lon, String city, String state_name, String postId) {
        this.lat = lat;
        this.lon = lon;
        this.city = city;
        this.state_name = state_name;
        this.postId = postId;
    }
}

class DistanceUtility {

    HashMap<String, ZipCodeData> zipCodesMap = new HashMap<String, ZipCodeData>();

    /**
     * Initialize zip codes using given file
     *
     * @throws IOException
     * @throws NumberFormatException
     */
    public DistanceUtility() throws NumberFormatException, IOException {

        BufferedReader br = new BufferedReader(new FileReader("zipCodePosId.csv"));

        String line = null;

        while ((line = br.readLine()) != null) {
            String str[] = line.split(",");

            String zipCode = str[0];

            double lat = Double.parseDouble(str[1]);
            double lon = Double.parseDouble(str[2]);
            ;
            String city = str[3];
            String state_name = str[4];
            String postId = str[5];

            ZipCodeData zipCodeData = new ZipCodeData(lat, lon, city, state_name, postId);
```

```
            zipCodesMap.put(zipCode, zipCodeData);
        }
        br.close();
    }

    /**
     *
     * @param zipcode1 - zip code of previous transaction
     * @param zipcode2 - zip code of current transaction
     * @return distance between two zip codes
     */
    public double getDistanceViaZipCode(String zipcode1, String zipcode2) {
        ZipCodeData z1 = zipCodesMap.get(zipcode1);
        ZipCodeData z2 = zipCodesMap.get(zipcode2);
        return distance(z1.lat, z1.lon, z2.lat, z2.lon);
    }

    private double distance(double lat1, double lon1, double lat2, double lon2) {
        double theta = lon1 - lon2;
        double dist = Math.sin(deg2rad(lat1)) * Math.sin(deg2rad(lat2))
                + Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) * Math.cos(deg2rad(theta));
        dist = Math.acos(dist);
        dist = rad2deg(dist);
        dist = dist * 60 * 1.1515;
        dist = dist * 1.609344;

        return dist;
    }

    private double rad2deg(double rad) {
        return rad * 180.0 / Math.PI;
    }

    private double deg2rad(double deg) {
        return deg * Math.PI / 180.0;
    }

}

/*
 * //To get the distance between two zipcodes from the main class public class
 * PostcodeCalculator { public static void main(String args[]) throws
 * NumberFormatException, IOException { DistanceUtility disUtil=new
 * DistanceUtility();
 *
 * System.out.println(disUtil.getDistanceViaZipCode("10001", "10524")); } }
 */
```

2. KafkaConsumer.java

This is the main program which connects with Kafka and fetch POS transactions which are in JSON format. Main logic was provided already in resources section of capstone project and used as is with few modifications to invoke logic for Credit Card Fraud Detection.

KafkaConsumer has main method which first setup SparkConf and then invokes JavaStreamingContext. This then setup context with SparkConf and batch duration of 1 second for the incoming D-Streams.

The application subscribes to Kafka stream as consumer for topic - **transactions-topic-verified**. 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 given problem statement for Credit Card Fraud Detection.

For the DStreams having Kafka topics, we are interested only in Value part so first fetch Value part by using Map transformation.

Once transactions are obtained, print number of transactions fetched and print all transactions fetched from Kafka topic.

Data from input stream is then mapped to the constructor of CreditCardFraudDetection class by using Map transformation.

After invoking CreditCardFraudDetection class constructor, each record in the JavaDStream becomes an object of type CreditCardFraudDetection and then invokes method in CreditCardFraudDetection class namely FraudDetection in order to classify transaction into GENUINE/FRAUD and update NoSQL DB.

Print Start time, start streaming and wait for termination. Upon receiving Ctrl+C response, gracefully close streaming and print End time.

Keep on monitoring the output file, few minutes after all transactions are processed probably in 3-4 minutes, Ctrl+C can be pressed to stop the program.

Below are the screenshots of KafkaConsumer class:

```java
package com.upgrad.creditcardfrauddetection;

import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Collection;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.VoidFunction;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka010.ConsumerStrategies;
import org.apache.spark.streaming.kafka010.KafkaUtils;
import org.apache.spark.streaming.kafka010.LocationStrategies;

public class KafkaConsumer {

    /*
     * Setup GROUP_ID in such a way that unique every time
     */
    public static String GROUP_ID = "amitgoelkafkaspark"
            + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());

    public static void main(String[] args) throws Exception {

        /*
         * Check if 1 argument is passed to the program
         */
        if (args.length != 1) {
            System.out.println("Please enter 1st argument as host server IP");
            return;
        }
        /*
         * Print GROUP_ID for current Kafka stream
         */
        System.out.println("Using group ID : " + GROUP_ID + "  for current Kafka stream.");

        /*
         * Set Logger to OFF
         */
        Logger.getLogger("org").setLevel(Level.OFF);
        Logger.getLogger("akka").setLevel(Level.OFF);

        /*
         * Set Spark Configuration object
         */
        SparkConf sparkConf = new SparkConf().setAppName("CreditCardFraudDetection").setMaster("local[*]");
```

```java
/*
 * Set Streaming Context
 */
JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, Durations.seconds(1));

/*
 * Initialize static variable hostServerIP in CreditCardFraudDetection class.
 * The HBase master and ZooKeeper are dependent on this IP, which keeps on
 * changing with restart of EC2 instance every time.
 */
CreditCardFraudDetection.hostServerIP = args[0];

/*
 * Set parameters for Kafka stream
 */
Map<String, Object> kafkaParams = new HashMap<>();
kafkaParams.put("bootstrap.servers", "100.24.223.181:9092");
kafkaParams.put("key.deserializer", StringDeserializer.class);
kafkaParams.put("value.deserializer", StringDeserializer.class);
kafkaParams.put("group.id", GROUP_ID);
kafkaParams.put("auto.offset.reset", "earliest");
kafkaParams.put("enable.auto.commit", true);

/*
 * Subscribe to Kafka topic
 */
Collection<String> topics = Arrays.asList("transactions-topic-verified");

/*
 * Consume input Kafka Stream and convert into Spark DStreams
 */
JavaInputDStream<ConsumerRecord<String, String>> stream = KafkaUtils.createDirectStream(jssc,
        LocationStrategies.PreferConsistent(),
        ConsumerStrategies.<String, String>Subscribe(topics, kafkaParams));

/*
 * Use map transformation on input stream and get the value part
 */
JavaDStream<String> jds = stream.map(x -> x.value());

/*
 * Print count of records present
 */
jds.foreachRDD(x -> System.out.println("Total Number of Transactions to be Processed : " + x.count() + "\n"));

/*
 * Print all the data so can be used for verification later if needed
 */
jds.foreachRDD(new VoidFunction<JavaRDD<String>>() {

    private static final long serialVersionUID = 1L;

    @Override
    public void call(JavaRDD<String> rdd) {

        rdd.foreach(a -> System.out.println(a));
    }
});
```

```
        /*
         * Map data from input stream to CreditCardFraudDetection class constructor
         */
        JavaDStream<CreditCardFraudDetection> jds_mapped = jds.map(x -> new CreditCardFraudDetection(x));

        /*
         * Call FraudDetection method in CreditCardFraudDetection class
         */
        jds_mapped.foreachRDD(new VoidFunction<JavaRDD<CreditCardFraudDetection>>() {

            private static final long serialVersionUID = 1L;

            @Override
            public void call(JavaRDD<CreditCardFraudDetection> rdd) {
                rdd.foreach(x -> x.FraudDetection(x));
            }

        });

        /*
         * Print current time stamp before starting
         */
        System.out.println("\nStart Time : " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()) + "\n");

        /*
         * Start Spark Streaming
         */
        jssc.start();

        /*
         * Await Termination to respond to Ctrl+C and gracefully close Spark Streaming
         */
        jssc.awaitTermination();

        /*
         * Print current time stamp before closing
         */
        System.out.println("\nEnd Time : " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()) + "\n");

        /*
         * Close Spark Streaming
         */
        jssc.close();

    }
}
```

3. CreditCardFraudDetection.java

This class is having business logic to validate incoming transaction against given validation rules and classify into GENUINE/FRAUD. This also updates NoSQL DB with details of current transaction.

This class has a **constructor** which is invoked by main method of KafkaConsumer class and each incoming transaction gets mapped to object of this class. It initializes member variables to values derived after parsing JSON from incoming transaction.

```java
public class CreditCardFraudDetection implements java.io.Serializable {

    /*
     * Declare member variables for this class
     */
    private String card_id = null;
    private String member_id = null;
    private double amount = 0.0;
    private String postcode = null;
    private String pos_id = null;
    private Date transaction_dt = null;
    private String status = null;

    private static Admin hBaseAdmin = null;
    public static String hostServerIP = null;

    /*
     * Default constructor defined for this class
     */
    public CreditCardFraudDetection() {
    }

    /*
     * Parameterized constructor for this class. It takes the incoming record from
     * DStream as JSON and parse it. This constructor initializes member variables
     * to the values derived from parsing the JSON.
     */
    public CreditCardFraudDetection(String jsonObj) {

        JSONParser parser = new JSONParser();
        try {

            JSONObject obj = (JSONObject) parser.parse(jsonObj);

            this.card_id = String.valueOf(obj.get("card_id"));
            this.member_id = String.valueOf(obj.get("member_id"));
            this.amount = Double.parseDouble(String.valueOf(obj.get("amount")));
            this.postcode = String.valueOf(obj.get("postcode"));
            this.pos_id = String.valueOf(obj.get("pos_id"));
            this.transaction_dt = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss")
                    .parse(String.valueOf(obj.get("transaction_dt")));

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

This class has a method namely **getHbaseAdmin** for setting up HBase connection. Server IP is passed as command line argument and initialized by main method of KafkaConsumer class. **Hard coding of server IP address is avoided here because public DNS keeps on changing for ec2 instance with every restart.**

```java
/*
 * getHbaseAdmin method for setting up the HBase connection. hostServerIP is
 * passed as command line argument and initialized by main method of
 * KafkaConsumer class.
 */
public static Admin getHbaseAdmin() throws IOException {

    Configuration conf = HBaseConfiguration.create();
    conf.setInt("timeout", 120000);
    conf.set("hbase.master", hostServerIP + ":60000");
    conf.set("hbase.zookeeper.quorum", hostServerIP);
    conf.set("hbase.zookeeper.property.clientPort", "2181");
    conf.set("zookeeper.znode.parent", "/hbase");
    try {
        Connection con = ConnectionFactory.createConnection(conf);
        if (hBaseAdmin == null) {
            hBaseAdmin = con.getAdmin();
        }

    } catch (IOException e) {
        e.printStackTrace();
    }

    return hBaseAdmin;

}
```

Main method of CreditCardFraudDetection class is **FraudDetection**. This is the method which is called for every incoming transaction. This method is called by main method of KafkaConsumer class.

It first classifies incoming transaction through classifyTransaction method into GENUINE/FRAUD. It then updates HBase tables by calling updateNoSQLDB method.

It also displays start time when processing of incoming transaction has started and end time when processing of transaction has ended.

```
/*
 * This is the method called by DStream records. It first classify transaction
 * through classifyTransaction method. It then calls updateNoSQLDB method to
 * update data in HBase tables.
 */
public void FraudDetection(CreditCardFraudDetection obj) {

    try {

        System.out.println("\n\n================================================================================");
        /*
         * Print time when transaction processing started
         */
        System.out.println("\nNew Transaction Processing Start Time : "
                + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()));

        System.out.println("\n======== Classify Current Transaction into GENUINE/FRAUD ==========");

        /*
         * classifyTransaction method to validate transaction against various rules and
         * classify current transaction into GENUINE/FRAUD
         */
        classifyTransaction();

        System.out.println("\n========== Update NoSQL Database for Current Transaction ==========");

        /*
         * updateNoSQLDB method to update NoSQL DB with card transaction details and if
         * transaction is GENUINE, update lookup table as well
         */
        updateNoSQLDB(obj);

        System.out.println("\n================================================================================");
        /*
         * Print time when transaction processing finished
         */
        System.out.println("\nCurrent Transaction Processing End Time : "
                + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()));

        System.out.println("\n================================================================================");

    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Method **classifyTransaction** called by FraudDetection method, apply provided 3 validation rules on each incoming transaction:

1. If credit score of the member is less than 200, then member could be a defaulter and it could be a possible case of fraud.

2. If transaction amount for incoming record/transaction, exceeds UCL (upper control limit) based on last 10 genuine transactions then it could be a possible case of fraud.

3. If speed of travel based on distance between postcode of current and last transaction, is more than 1km in 4sec = 0.25 km/sec then it could be a possible case of fraud

This method first finds out UCL for Card ID from HBase lookup table by calling method getUCL and displays UCL and current transaction amount. Then method finds out credit score of member owning Card ID from HBase lookup table by calling method getScore and displays credit score. Then method calls getSpeed method to find out speed at which member could have travelled based on distance between post code for current transaction and post code for last GENUINE transaction.

Explanation so far is shown in screenshot below:

```java
/*
 * Classify incoming transaction at POS as GENUINE or FRAUD
 */
private void classifyTransaction() throws IOException {

    /*
     * Lets apply provided rules on each transaction:
     *
     * 1. If member score is less than 200, then member could be a defaulter and
     * possible case of fraud.
     *
     * 2. If transaction amount for current transaction, exceeds UCL (upper control
     * limit) based on last 10 genuine transactions then possible case of fraud.
     *
     * 3. If speed of travel based on distance between postcode of current and last
     * transaction, is more than 1km in 4sec = 0.25 km/sec then possible case of
     * fraud.
     */

    double card_ucl = 0;
    int card_score = 0;
    double speed = 0;

    /*
     * Print Card ID of current transaction
     */
    System.out.println("\nCard ID of current transaction is : " + this.getCard_id());

    /*
     * Get UCL for Card ID
     */
    card_ucl = getUCL(this.getCard_id());
    System.out.println("\nUCL (Upper Control Limit) for Card ID of current transaction is : " + card_ucl);

    /*
     * Print current transaction amount
     */
    System.out.println("\nCurrent Transaction Amount is : " + this.getAmount());

    /*
     * Get score for Card ID
     */
    card_score = getScore(this.getCard_id());
    System.out.println("\nCredit score for Card ID of current transaction is : " + card_score);

    /*
     * Get speed of current transaction in relation to last transaction
     */
    speed = getSpeed();
```

Method then checks if credit score >= 200 and amount of current transaction is <= UCL and speed is <= 0.25 and if so then status of the transaction is classified as ‘GENUINE’ as it has passed all 3 validation rules. Details are displayed accordingly.

If any of the validation rules fail then status of the transaction is classified as ‘FRAUD’ and failed validation rule is displayed for the information.

Explanation so far is shown in screenshot below:

```
        /*
         * All 3 rules need to be passed for card transaction status to be GENUINE
         */
        if ((card_score >= 200) && (this.getAmount() <= card_ucl) && (speed <= 0.25)) {
            this.status = "GENUINE";
            System.out.println("\nCurrent Card Transaction has passed Credit Score Validation rule : Score : "
                    + card_score + " >= 200");
            System.out.println(
                    "\nCurrent Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : "
                            + this.getAmount() + " <= " + card_ucl);
            System.out.println(
                    "\nCurrent Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : "
                            + speed + " <= 0.25");
            System.out.println("\nCurrent Card Transaction has passed all 3 Validation rules and so status is : "
                    + this.getStatus());
        } else {
            this.status = "FRAUD";
            if (card_score < 200) {
                System.out.println("\nCurrent Card Transaction did not pass through Credit Score Validation rule");
            }
            if (speed > 0.25) {
                System.out.println("\nCurrent Card Transaction did not pass through Zip Code Distance Validation rule");
            }
            if (this.getAmount() > card_ucl) {
                System.out.println(
                        "\nCurrent Card Transaction did not pass through UCL (Upper Control Limit) Validation rule");
            }
            System.out.println("\nCurrent Card Transaction status is : " + this.getStatus());
        }
    }
```

Method **getUCL** called by classifyTransaction method, first establishes HBase connection if not established already by calling getHbaseAdmin method. It then fetches UCL for input Card ID from lookup_data_hive lookup HBase table. This method code is shown in screenshot below:

```
/*
 * getUCL method to look up upper control limit for input Card ID from
 * lookup_data_hive HBASE table
 */
public static Double getUCL(String cardID) throws IOException {

    if (cardID == null) {
        System.out.println("\nCard ID is not present in data recieved from Kafka. Kindly check Kafka stream");
        return -1d;
    } else {

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

        /*
         * Setup table name, get data for input card id and get ucl from result set. ucl
         * belongs to lookup_card_family column family in look up table
         * lookup_data_hive.
         */
        String table = "lookup_data_hive";
        double val = 0;
        try {
            Get cardId = new Get(Bytes.toBytes(cardID));
            Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(table));
            Result result = htable.get(cardId);
            byte[] value = result.getValue(Bytes.toBytes("lookup_card_family"), Bytes.toBytes("ucl"));
            if (value != null) {
                val = Double.parseDouble(Bytes.toString(value));
            } else {
                val = 0d;
            }

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

        return val;
    }
}
```

Method **getScore** called by classifyTransaction method, first establishes HBase connection if not established already by calling getHbaseAdmin method. It then fetches credit score for input Card ID from lookup_data_hive lookup HBase table. This method code is shown in screenshot below:

```
/*
 * getScore method to look up score for input Card ID from lookup_data_hive
 * HBASE table
 */
public static Integer getScore(String cardID) throws IOException {

    if (cardID == null) {
        System.out.println("\nCard ID is not present in data recieved from Kafka. Kindly check Kafka stream");
        return -1;
    } else {

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

        /*
         * Setup table name, get data for input card id and get score from result set.
         * score belongs to lookup_card_family column family in look up table
         * lookup_data_hive.
         */
        String table = "lookup_data_hive";
        int val = 0;
        try {
            Get cardId = new Get(Bytes.toBytes(cardID));
            Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(table));
            Result result = htable.get(cardId);
            byte[] value = result.getValue(Bytes.toBytes("lookup_card_family"), Bytes.toBytes("score"));
            if (value != null) {
                val = Integer.parseInt(Bytes.toString(value));
            } else
                val = 0;

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

        return val;
    }
}
```

Method **getSpeed** called by classifyTransaction method, first creates an object of DistanceUtility class. It then fetches post code of last genuine transaction for input Card ID from HBase lookup table by calling getPostCode method and display post code of both last genuine and current transaction. It then fetches transaction date of last genuine transaction for input Card ID from HBase lookup table by calling getTransactionDate method and display transaction date of both last genuine and current transaction. It then calls getDistanceViaZipCode method on object of DistanceUtility class by passing post code of last genuine and current transaction. This call returns distance between 2 post codes in kilo meters (Km). Method then calculates time difference between transaction date of last genuine and current transaction which comes out in milli second and divide by 1000 to get in seconds. Speed is then calculated by dividing distance with time difference. Speed = Distance / Time. Distance between post code of last genuine and current transaction is divided by time difference between transaction date of last genuine and current transaction to obtain speed which comes out in km/sec. This is then returned to calling method.

Explanation so far is shown in screenshots below:

```java
/*
 * getSpeed method to calculate speed in km/sec based on distance between post
 * code of current and last transaction. This method makes of DistanceUtility
 * class provided
 */
private double getSpeed() {

    double distance = 0;
    long timeDifference = 0L;
    double speed = 0;

    try {

        /*
         * Create an object of DistanceUtility class
         */
        DistanceUtility distUtil = new DistanceUtility();

        /*
         * Get last post code of Card ID from lookup table in HBase
         */
        String lastPostCode = getPostCode(this.getCard_id());
        System.out.println("\nLast Post Code for Card ID of current transaction is : " + lastPostCode);

        /*
         * Print post code of current transaction
         */
        System.out.println("\nCurrent Post Code for Card ID of current transaction is : " + this.getPostcode());

        /*
         * Get last transaction_dt of Card ID from Lookup table in HBase
         */
        Date lastTransactionDt = getTransactionDate(this.getCard_id());
        System.out.println("\nLast Transaction Date for Card ID of current transaction is : "
                + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(lastTransactionDt));

        /*
         * Print transaction date of current transaction
         */
        System.out.println("\nCurrent Transaction Date for Card ID of current transaction is : "
                + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(this.getTransaction_dt()));

        /*
         * Get distance between 2 post code, calculated by following method in
         * DistanceUtility class. Distance is in kilometers.
         */
        distance = distUtil.getDistanceViaZipCode(lastPostCode, this.getPostcode());
        System.out
                .println("\nDistance (returned by DistanceUtility) between current and last postcode (in km) is : "
                        + distance);
```

```
        /*
         * Time difference between 2 transaction dates is calculated as below. The
         * difference is returned in milliseconds so dividing by 1000 to get time in
         * seconds. For many transactions, the incoming transaction date is less than
         * last transaction date stored in lookup table so using abs method.
         */
        timeDifference = (java.lang.Math.abs(this.getTransaction_dt().getTime() - lastTransactionDt.getTime()))
                / 1000;
        System.out.println("\nAbsolute Time Difference between current and last transaction (in seconds) is : "
                + timeDifference);

        /*
         * Speed is being calculated in km/sec
         *
         * At one instance, time difference was zero (2018-06-21 15:29:59) and last and
         * current post code (55311) were same, distance utility returned non zero value
         * (9.4930730546311141E-5) which is quite small as such but still a non zero
         * value was returned, due to which speed came out to be infinity for card_id =
         * '4838289241690162'. So added this check if last and current post code are
         * same then consider speed as 0.
         */

        if (lastPostCode.equals(this.getPostcode())) {
            System.out.println(
                    "\nSince Last Post Code and Current Post Code are same, so distance is 0 and so considering speed as 0");
            speed = 0;
        } else {
            speed = distance / timeDifference;
        }

        System.out.println("\nSpeed (Distance / Time) : " + speed + " km/second ");

    } catch (Exception e) {

        e.printStackTrace();
    }

    return speed;
}
```

==========

If post code of last genuine transaction and current transaction is same then distance is considered as 0. There is a problem with DistanceUtility provided that it returns non-zero value at times when post code of last genuine transaction and current transaction is same. That is not ideal so considering distance as 0 in such cases to avoid this problem.

I have provided Output.txt as one of the artifact. Put this in a directory on ec2 instance and run this unix command:

grep "Distance (returned by DistanceUtility) between current and last postcode (in km) is : .*E-.*" Output.txt

Below output will be returned by the command:

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 1.3425232662457948E-4
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 1.3425232662457948E-4
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5

These are all cases where DistanceUtility returned non zero value while should have returned zero because post code of last genuine and current transaction is same.

Below are some more details fetched from Output.txt for these cases:

Card ID of current transaction is : 4838289241690162
Last Post Code for Card ID of current transaction is : 55311
Current Post Code for Card ID of current transaction is : 55311
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5

Card ID of current transaction is : 371829395919892
Last Post Code for Card ID of current transaction is : 60929
Current Post Code for Card ID of current transaction is : 60929
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5

Card ID of current transaction is : 4750699680073601
Last Post Code for Card ID of current transaction is : 88339
Current Post Code for Card ID of current transaction is : 88339
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5

Card ID of current transaction is : 4552141762363568
Last Post Code for Card ID of current transaction is : 55735
Current Post Code for Card ID of current transaction is : 55735
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 1.3425232662457948E-4

Card ID of current transaction is : 344997383137673
Last Post Code for Card ID of current transaction is : 89043
Current Post Code for Card ID of current transaction is : 89043
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 1.3425232662457948E-4

Card ID of current transaction is : 344250415187512
Last Post Code for Card ID of current transaction is : 97015
Current Post Code for Card ID of current transaction is : 97015
Distance (returned by DistanceUtility) between current and last postcode (in km) is : 9.493073054631141E-5

==========

Method **getPostCode** called by getSpeed method, first establishes HBase connection if not established already by calling getHbaseAdmin method. It then fetches post code for input Card ID from lookup_data_hive lookup HBase table. This method code is shown in screenshot below:

```
/*
 * getPostCode method to look up last post code for input Card ID from
 * lookup_data_hive HBASE table
 */
public static String getPostCode(String cardID) throws IOException {

    if (cardID == null) {
        System.out.println("\nCard ID is not present in data recieved from Kafka. Kindly check Kafka stream");
        return null;
    } else {

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

        /*
         * Setup table name, get data for input card id and get postcode from result
         * set. postcode belongs to lookup_transaction_family column family in look up
         * table lookup_data_hive.
         */
        String table = "lookup_data_hive";
        String val = null;
        try {
            Get cardId = new Get(Bytes.toBytes(cardID));
            Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(table));
            Result result = htable.get(cardId);
            byte[] value = result.getValue(Bytes.toBytes("lookup_transaction_family"), Bytes.toBytes("postcode"));
            if (value != null) {
                val = Bytes.toString(value);
            } else
                val = null;

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

        return val;
    }
}
```

Method **getTransactionDate** called by getSpeed method, first establishes HBase connection if not established already by calling getHbaseAdmin method. It then fetches transaction date for input Card ID from lookup_data_hive lookup HBase table. This method code is shown in screenshot below:

```java
/*
 * getTransactionDate method to look up last transaction date for input Card ID
 * from lookup_data_hive HBASE table
 */
public static Date getTransactionDate(String cardID) throws IOException {

    if (cardID == null) {
        System.out.println("\nCard ID is not present in data recieved from Kafka. Kindly check Kafka stream");
        return null;
    } else {

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

        /*
         * Setup table name, get data for input card id and get transaction_dt from
         * result set. transaction_dt belongs to lookup_transaction_family column family
         * in look up table lookup_data_hive.
         */
        String table = "lookup_data_hive";
        Date val = null;
        try {
            Get cardId = new Get(Bytes.toBytes(cardID));
            Table htable = hBaseAdmin.getConnection().getTable(TableName.valueOf(table));
            Result result = htable.get(cardId);
            byte[] value = result.getValue(Bytes.toBytes("lookup_transaction_family"),
                    Bytes.toBytes("transaction_dt"));
            if (value != null) {
                val = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(Bytes.toString(value));
            } else
                val = null;

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

Method updateNoSQLDB called by FraudDetection method in CreditCardFraudDetection class accomplishes Task 6 and Task 7 from the given problem statement.

It performs 2 activities:

1. It inserts incoming card transaction details in card_transactions_hive HBase table. **(Task 6)**

   It first establishes HBase connection if not established already by calling getHbaseAdmin method. It converts transaction date of incoming transaction in a suitable format for HBase table. Row key of HBase table is populated using randomUUID() so method uses randomUUID() as row key for the HBase table. New Put object is created using row key and then other columns are added to it like card_id, member_id, amount, postcode, pos_id, transaction_dt and status. These are all then saved down in card_transactions_hive HBase table. If by chance, HBase table does not exist then appropriate message gets displayed accordingly.

2. It also updates lookup_data_hive lookup HBase table with current post code and transaction date if and only if the status of current transaction is GENUINE. **(Task 7)**

   It first establishes HBase connection if not established already by calling getHbaseAdmin method. Card ID is used as row key for lookup HBase table. New Put object is created using row key and then transaction date (in a suitable format for HBase table) and post code are added to it. These are all then saved down in lookup_data_hive lookup HBase table. If by chance, HBase lookup table does not exist then appropriate message gets displayed accordingly.

Explanation so far is shown in screenshots below:

```java
/*
 * updateNoSQLDB method performs 2 activities:
 *
 * 1. It inserts incoming card transaction details in card_transactions_hive
 * HBASE table.
 *
 * 2. It also updates lookup_data_hive HBASE table with current post code and
 * transaction date if and only if the status of current transaction is GENUINE.
 */
public static void updateNoSQLDB(CreditCardFraudDetection transactionData) throws IOException {

    try {

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

        /*
         * Print Card ID of current transaction
         */
        System.out.println("\nCard ID of current transaction is : " + transactionData.getCard_id());

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

        Double Amount = transactionData.getAmount();

        String transactionTable = "card_transactions_hive";

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

            /*
             * Use randomUUID as row key
             */

            UUID transactionID = UUID.randomUUID();

            Put p = new Put(Bytes.toBytes(transactionID.toString()));
```

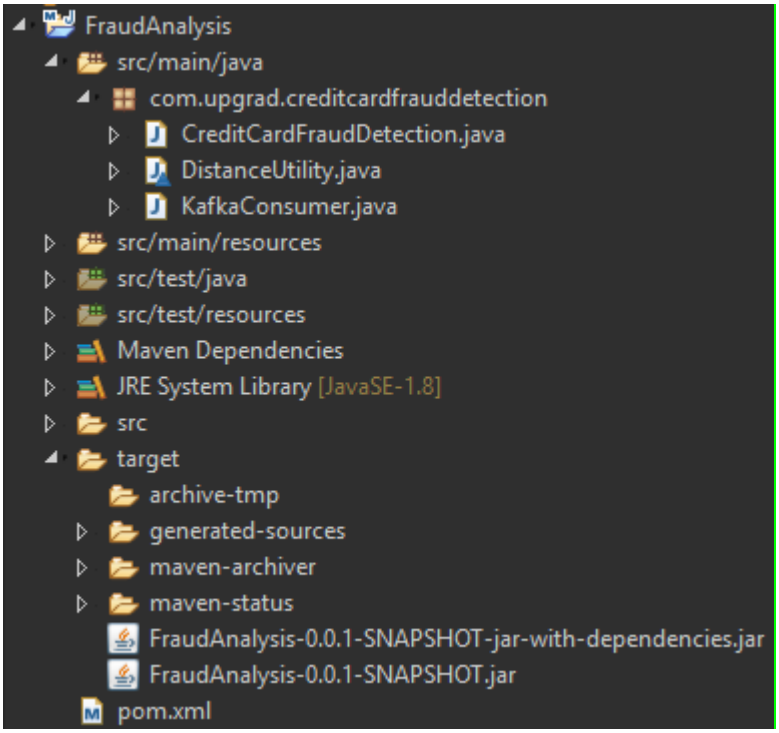
```
            /*
             * Add column values for each column
             */
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("card_id"),
                    Bytes.toBytes(transactionData.getCard_id()));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("member_id"),
                    Bytes.toBytes(transactionData.getMember_id()));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("amount"),
                    Bytes.toBytes(Amount.toString()));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("postcode"),
                    Bytes.toBytes(transactionData.getPostcode()));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("pos_id"),
                    Bytes.toBytes(transactionData.getPos_id()));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("transaction_dt"),
                    Bytes.toBytes(val));
            p.addColumn(Bytes.toBytes("card_transactions_family"), Bytes.toBytes("status"),
                    Bytes.toBytes(transactionData.getStatus()));

            htable.put(p);
            System.out.println("\nCurrent Transaction Details are populated in Card Transactions HBase Table : "
                    + transactionTable);
        } else {
            System.out.println("\nHBase Table named : " + transactionTable + " : does not exist");
        }

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

    /*
     * Check if status is Genuine to update lookup_data_hive HBase table
     */
    if (transactionData.getStatus().equals("GENUINE")) {

        System.out.println("\nCurrent Card Transaction status is GENUINE so updating lookup table");

        try {

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

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

            String lookupTable = "lookup_data_hive";

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

```
                /*
                 * Instantiate Put class to Card ID row key in lookup_data_hive table
                 */
                Put p = new Put(Bytes.toBytes(transactionData.getCard_id()));

                /*
                 * Add column values for postcode and transaction_dt
                 */
                p.addColumn(Bytes.toBytes("lookup_transaction_family"), Bytes.toBytes("postcode"),
                        Bytes.toBytes(transactionData.getPostcode()));
                p.addColumn(Bytes.toBytes("lookup_transaction_family"), Bytes.toBytes("transaction_dt"),
                        Bytes.toBytes(val));

                /*
                 * Save put instance to HTable.
                 */
                htable.put(p);
                System.out.println(
                        "\nPostcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : "
                                + lookupTable);

            } else {
                System.out.println("\nHBase Lookup Table named : " + lookupTable + " : does not exist");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("\nCurrent Card Transaction status is FRAUD so not updating lookup table");
    }

}
```

========== Logic Explanation for Task 5 - 6 -7 – END ==========

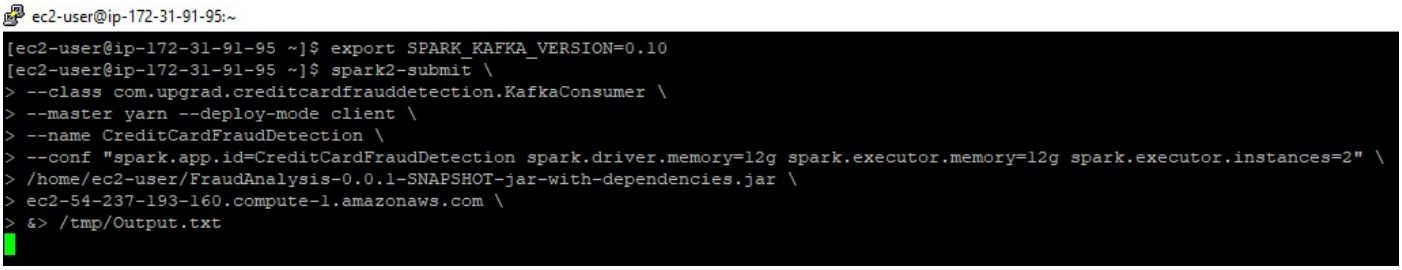
========== Code Execution – Start ==========

FraudAnalysis.zip has FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar as well as maven project. Either provided jar can be used or maven project can be imported in eclipse and fresh jar can be generated in eclipse. After project is imported, ensure java build path is pointing to jdk 1.8 and project compliance is also set to 1.8. Do maven clean followed by maven install. It will generate fresh jar with same name.



Copy FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar and zipCodePosId.csv in /home/ec2-user directory of ec2 instance and run below commands to execute the program:

```
export SPARK_KAFKA_VERSION=0.10

spark2-submit \
--class com.upgrad.creditcardfrauddetection.KafkaConsumer \
--master yarn --deploy-mode client \
--name CreditCardFraudDetection \
--conf "spark.app.id=CreditCardFraudDetection spark.driver.memory=12g spark.executor.memory=12g
spark.executor.instances=2" \
/home/ec2-user/FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
ec2-54-237-193-160.compute-1.amazonaws.com \
&> /tmp/Output.txt
```



Open another session for ec2 instance and run:

```
tail -f /tmp/Output.txt
```

ec2-user@ip-172-31-91-95:/tmp

```
Current Card Transaction has passed Credit Score Validation rule : Score : 683 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 6880309.0 <= 1.0869782716383055E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 1.3972584550746266E-5 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 5391723993945313

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

============================================================

Current Transaction Processing End Time : 2019-06-01 11:15:53

============================================================
Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0
```

Once all transactions are processed, press Ctrl+C on the session where spark command was executed.

ec2-user@ip-172-31-91-95:~

```
[ec2-user@ip-172-31-91-95 ~]$ export SPARK_KAFKA_VERSION=0.10
[ec2-user@ip-172-31-91-95 ~]$ spark2-submit \
> --class com.upgrad.creditcardfrauddetection.KafkaConsumer \
> --master yarn --deploy-mode client \
> --name CreditCardFraudDetection \
> --conf "spark.app.id=CreditCardFraudDetection spark.driver.memory=12g spark.executor.memory=12g spark.executor.instances=2" \
> /home/ec2-user/FraudAnalysis-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
> ec2-54-237-193-160.compute-1.amazonaws.com \
> &> /tmp/Output.txt
^C[ec2-user@ip-172-31-91-95 ~]$
```

Below is the screenshot of other session of ec2 instance after program has ended.

ec2-user@ip-172-31-91-95:/tmp

```
Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0

Total Number of Transactions to be Processed : 0


End Time : 2019-06-01 11:17:22
```

Run below command to extract card_transactions_hbase table into a file:

```
hive -S -e 'use capstone_project; set hive.cli.print.header=true; select * from card_transactions_hbase' | sed 's/[\t]/,/g' | sed 's/card_transactions_hbase\.//g' | sed '/^WARN:/d' > /tmp/card_transactions.csv
```

Below are the files created in /tmp directory

ec2-user@ip-172-31-91-95:/tmp

```
[ec2-user@ip-172-31-91-95 tmp]$ ls -lrt card_transactions.csv Output.txt
-rw-rw-r-- 1 ec2-user ec2-user 13157831 Jun  1 11:17 Output.txt
-rw-rw-r-- 1 ec2-user ec2-user  7693571 Jun  1 11:25 card_transactions.csv
[ec2-user@ip-172-31-91-95 tmp]$
```

Below are no of transactions classified into GENUINE and FRAUD in final output card_transactions.csv

Total: 59367

GENUINE: 59150
FRAUD: 217

ec2-user@ip-172-31-91-95:/tmp

```
[ec2-user@ip-172-31-91-95 tmp]$ grep GENUINE card_transactions.csv | wc -l
59150
[ec2-user@ip-172-31-91-95 tmp]$ grep FRAUD card_transactions.csv | wc -l
217
[ec2-user@ip-172-31-91-95 tmp]$
```

Below are no of transactions provided in original card_transactions.csv

Total: 53292

GENUINE: 53210
FRAUD: 82

ec2-user@ip-172-31-91-95:~/capstone_project

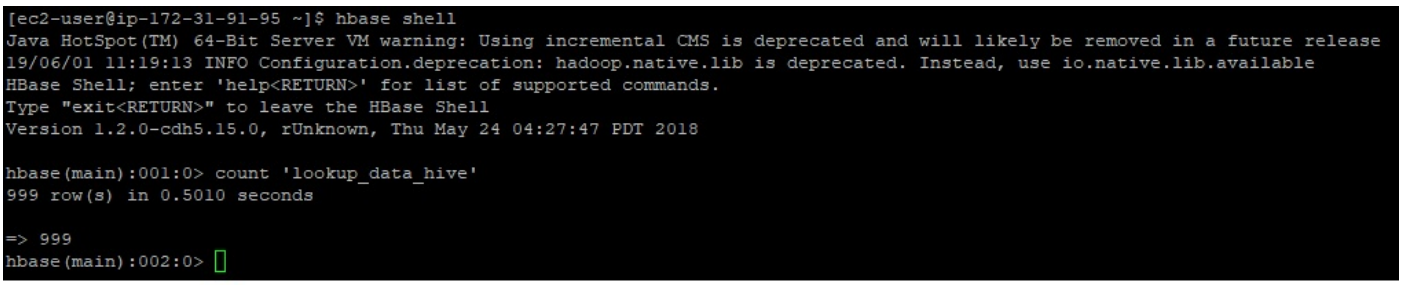
```
[ec2-user@ip-172-31-91-95 capstone_project]$ ls -lrt card_transactions.csv
-rw-rw-r-- 1 ec2-user ec2-user 4829520 May 25 05:04 card_transactions.csv
[ec2-user@ip-172-31-91-95 capstone_project]$ grep GENUINE card_transactions.csv |wc -l
53210
[ec2-user@ip-172-31-91-95 capstone_project]$ grep FRAUD card_transactions.csv |wc -l
82
[ec2-user@ip-172-31-91-95 capstone_project]$
```

========== Code Execution – END ==========

========== Output – Start ==========

Below are the screenshots from HBase after spark program was over:

Count of records in lookup_data_hive lookup table

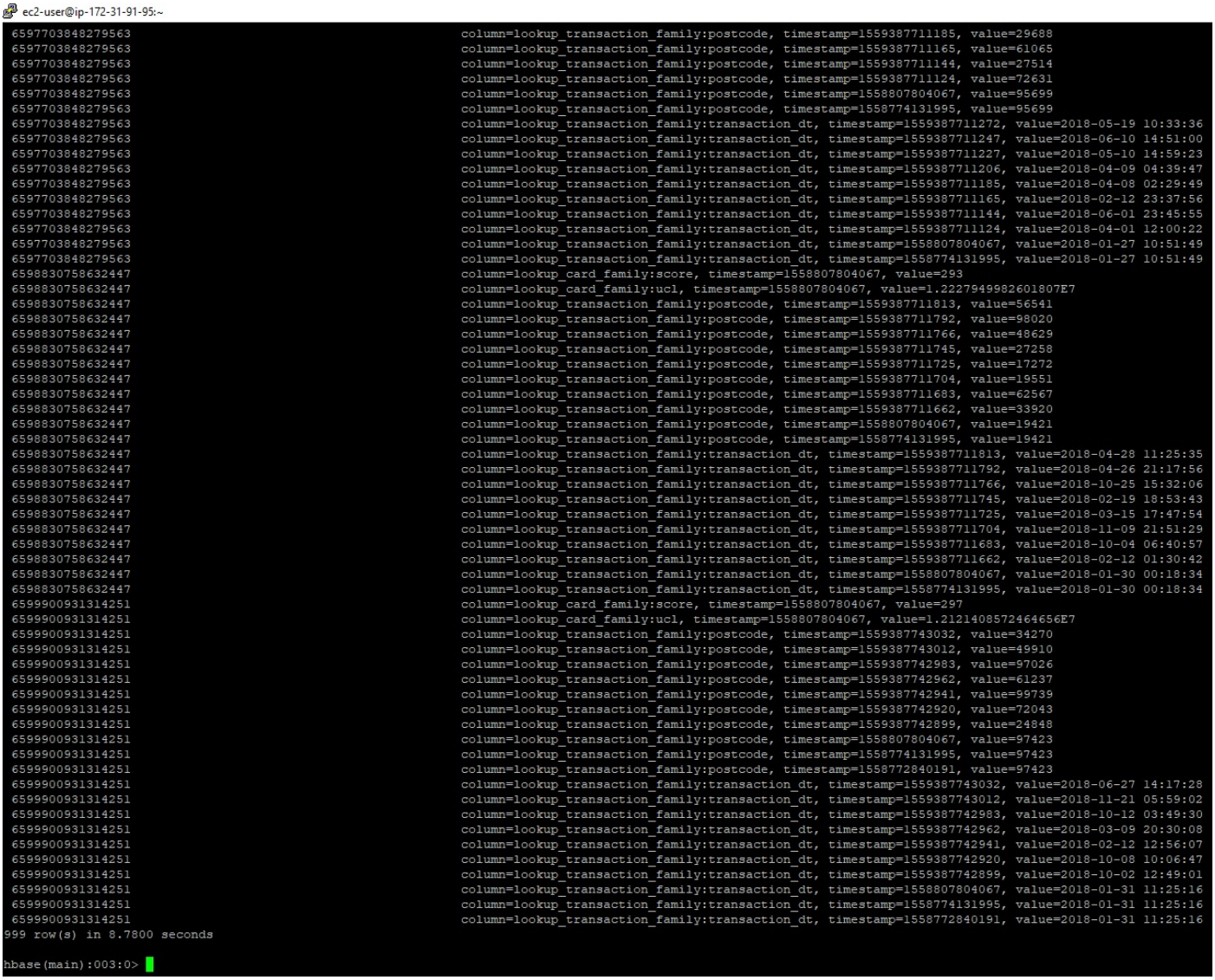


Count of records in card_transactions_hive table

ec2-user@ip-172-31-91-95:~

```
hbase(main):001:0> count 'card_transactions_hive'
Current count: 1000, row: 048516cc-b769-4cf8-bf42-cdc613d99982
Current count: 2000, row: 08c5579d-d6c8-4011-8928-76ddf315e899
Current count: 3000, row: 0cea0bfe-b105-455c-a5e1-3c9e0d165657
Current count: 4000, row: 10f5ea47-2985-4dd2-ae83-d0ce5eaffec1
Current count: 5000, row: 155dfa6e-01b8-4e80-8114-b6a0d450301e
Current count: 6000, row: 19d5e179-81e4-4a96-9118-44e974a6f0b3
Current count: 7000, row: 1dda85ff-0566-441b-bd10-b63c07fd4b98
Current count: 8000, row: 22406713-fcdb-4b6d-bf93-621dc697c4c4
Current count: 9000, row: 2688d737-eda4-4636-b73b-0ecaa517ad0d
Current count: 10000, row: 2acf7f89-9a47-4fbc-b054-c9616ldac4f8
Current count: 11000, row: 2f69d859-1bf3-4d6e-b0e1-bc86ab32d0b7
Current count: 12000, row: 338a91d2-852f-4187-86ad-af47af5f13a4
Current count: 13000, row: 37d046d8-7c74-4337-aeb4-a0988e06f3d0
Current count: 14000, row: 3c3922e0-2ecd-4419-b31d-62d16ad998a3
Current count: 15000, row: 40a6b2fb-255c-4540-a5b0-bf9361e1ce6e
Current count: 16000, row: 44f7e2f6-2e6b-4ada-9b57-97d558e1afc5
Current count: 17000, row: 492513ce-76e4-4d30-9ac7-32cb449f4f10
Current count: 18000, row: 4da0a031-e2a2-4234-9a04-fd8a0bf132ca
Current count: 19000, row: 51d191ee-fde4-4bad-ac0e-8bf061b682b2
Current count: 20000, row: 56494ff8-5ba9-43cd-a223-a53170c07ae8
Current count: 21000, row: 5a961c14-019d-4d25-b8d3-f927929b5f6d
Current count: 22000, row: 5f06439c-51f9-4da7-b51d-563868da8b5d
Current count: 23000, row: 63493804-2434-4d16-b978-2671b9a29c2b
Current count: 24000, row: 67975634-3bcc-4b74-92af-ea8d0006a25e
Current count: 25000, row: 6bd95cfa-b96b-465d-a792-fd17b2c9ed67
Current count: 26000, row: 7025b63f-c3d9-4d89-b02d-6641b34eedf5
Current count: 27000, row: 7445a166-a422-4203-8695-5a3c8f25090f
Current count: 28000, row: 788de720-7f17-4040-8bf4-6078cd54b78c
Current count: 29000, row: 7cee5c9b-70bd-4099-a632-c2962dbc5206
Current count: 30000, row: 8122c635-db39-4610-ac1a-cfb865669e92
Current count: 31000, row: 856966d4-c697-4635-a9c2-14c98c7c1ab3
Current count: 32000, row: 89f38c2a-a752-4f23-9b03-18021418caba
Current count: 33000, row: 8e66f3c7-61d9-4995-bd6f-eb36f150653e
Current count: 34000, row: 92c9e120-f7f5-4511-8338-37e64bc1ea12
Current count: 35000, row: 971bbd7e-feee-4d23-85e6-ed5a360fe769
Current count: 36000, row: 9b7b29a3-93ea-4206-ac44-1134886a5b0f
Current count: 37000, row: 9fdd08fc-5024-4703-b81a-4e8a7dc232c3
Current count: 38000, row: a40921fb-fc8c-47e6-a140-8fbf027e15a2
Current count: 39000, row: a815bb5d-d856-418b-a6a5-9a0af01cf499
Current count: 40000, row: ac73e814-be68-47a0-8e17-a56aa416b4aa
Current count: 41000, row: b0e11a94-cd8f-4e88-9765-0edcecc278e0
Current count: 42000, row: b5370f72-caf6-408d-99c0-a2f886fd533c
Current count: 43000, row: b9131fa3-1120-41d8-a08c-4a948d475d48
Current count: 44000, row: bd7f69fb-afdb-4aa9-bb02-1002f41bc9cb
Current count: 45000, row: c213bb6e-00eb-4cf1-9a0c-712e6c49af80
Current count: 46000, row: c652b30b-6af6-4040-9e3d-294d984c8f81
Current count: 47000, row: ca8f7a3b-032a-4ed3-91c2-bdc30fb88606
Current count: 48000, row: cf1101ca-b289-4598-a79e-fa70ce5ccb61
Current count: 49000, row: d3453dbd-499a-434f-a7a0-16166320289b
Current count: 50000, row: d7837c76-5aaa-4211-a6f8-5b1782c29ba2
Current count: 51000, row: dbed11ab-aa1c-421f-8748-90385bef2006
Current count: 52000, row: e050f84a-f79a-40e7-b8c3-ad8f213b6fca
Current count: 53000, row: e4c0e75c-8b59-4af6-9966-f4bc19bd0850
Current count: 54000, row: e9008eb4-df31-4aac-8152-8664820fde47
Current count: 55000, row: ed3d084f-caf5-49a9-9455-aed812e99b7c
Current count: 56000, row: f16f04e2-79f6-492c-bbaa-83158489f599
Current count: 57000, row: f5b59b60-081a-4466-8151-a1fd87502013
Current count: 58000, row: fa1cba61-fc9b-4ffd-8da3-ac07e04399d0
Current count: 59000, row: fe8b817a-1cb0-44ba-b2c7-49ab94043afa
59367 row(s) in 3.5890 seconds

=> 59367
```

Some records from lookup_data_hive lookup table. I had created 2 column families in it namely lookup_card_family and lookup_transaction_family. Column family lookup_card_family has columns ucl and score and can store up to 1 VERSION. Column family lookup_transaction_family has columns postcode and transaction_dt and can store up to 10 VERSIONS. Please refer to LogicMid.pdf for more details.



Below are the screenshots from Hive after spark program was over:

Below screenshot shows some records from card_transactions_hbase table

```
[ec2-user@ip-172-31-91-95 ~]$ hive
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
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> use capstone_project;
OK
Time taken: 1.757 seconds
hive> select * from card_transactions_hbase limit 20;
OK
00000d47-75fb-4b26-9b24-2c7b87db6ce2    5175055180166201        689714705796007 167127.0        45764   486276156459636 2017-11-27 08:56:20     GENUINE
0003e0a0-5544-4b1b-be7f-7081f2f14ed3    6451849404352750        366586364589156 3778904.0       54667   176540398942621 2017-02-08 22:38:07     GENUINE
00046135-56ad-41d4-bde8-3c51010c3547    343330859464824 051518771748227 5014013.0       81089   860541878277349 2016-02-04 07:01:07     GENUINE
0004d284-c1bc-4bc0-9ecc-d70bf6b78390    349254330876970 175914389419795 5887044.0       50544   725031279093414 2017-05-11 09:07:59     GENUINE
00063233-8ff7-40bc-a74a-47826c594d8a    377201318164757 924475891017022 6881323.0       95975   749826149212397 2018-07-28 06:40:31     GENUINE
0008d2a6-c996-4ee2-83f7-014851a7a6fb    4907253800863053        723132659200637 5488105.0       23117   617446577958996 2017-05-08 23:11:54     GENUINE
0008dedf-1f7c-432b-9ed0-d2ac1446284f    6011920623387365        421308864367180 6482829.0       99114   28654301191234  2018-12-02 10:22:18     GENUINE
0009d330-bd31-456b-bf6c-7ab5b1dfb4b2    4546430394425179        459011730675059 2788646.0       52720   150020495648797 2017-06-07 07:11:35     GENUINE
000a3754-f342-42cb-9976-2c9162d03402    6227994101600953        940184607999133 5002389.0       21555   972412969980550 2017-01-04 18:11:33     GENUINE
000b1f21-c498-4fd0-9e8b-bb9ed6f9ade2    3445834880345238 667772346348129 1982685.0       52254   608391261953205 2017-04-06 16:58:46     GENUINE
000bdfa1-3f51-4d8b-a9ee-2039ea5db1f4    3459492604347 68 317939112617073 8462321.0       12124   158131710971474 2017-04-29 14:28:54     GENUINE
000ebe74-8279-4de1-800a-03ce39620262    4250028739827574        394067902581867 5430837.0       33820   491369407491686 2017-09-07 09:29:30     GENUINE
00118671-5ae9-42ca-bfdb-a5d1e3d5dbb6    4482511371042362        665997088492282 6761452.0       49620   790484082532571 2017-10-26 13:15:13     GENUINE
00121669-4735-4967-95e2-d8616a04d472    6451387765835527        188977299867156 3086412.0       99761   028580296523857 2017-12-05 01:16:36     GENUINE
00124dc6-0526-44c8-8f27-98e5ac62fc5e    5381466904123396        550262364037847 1136800.0       58269   715722111477598 2017-02-09 17:56:48     GENUINE
001312a6-c93c-45cc-873d-0fc3cc6f3548    4750699680073601        677295725087734 1187814.0       51246   767313351253016 2017-07-20 07:29:06     GENUINE
00144543-b32c-4d5c-bce2-8a97e354eac8    347274385563543 359453922401698 2757804.0       97133   183518219701679 2016-03-15 23:07:53     GENUINE
00184dd8-2c9d-4304-ae89-76c71924fadb    5252551880815473        356544959463542 5199495.0       83101   025098568987945 2017-09-14 04:45:44     GENUINE
001854db-e5f2-4dc9-b9be-c7463a45008e    6011980249342475        278094439476476 4436433.0       30144   816262392083518 2017-12-05 04:47:17     GENUINE
0019695d-8a44-447c-94c6-f2c622713d54    3426799595520647 328980632419725 4342881.0       59867   641574580824598 2017-09-01 02:30:36     GENUINE
Time taken: 0.668 seconds, Fetched: 20 row(s)
hive>
```

Below screenshot shows count of records in card_transactions_hbase table

```
hive> select count(*) from card_transactions_hbase;
Query ID = ec2-user_20190601114545_57ab7485-a9b2-474c-88de-b07a6e378e03
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_1559386504130_0003, Tracking URL = http://ip-172-31-91-95.ec2.internal:8088/proxy/application_1559386504130_0003/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1559386504130_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-06-01 11:46:00,586 Stage-1 map = 0%,  reduce = 0%
2019-06-01 11:46:07,924 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.99 sec
2019-06-01 11:46:14,167 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.11 sec
MapReduce Total cumulative CPU time: 9 seconds 110 msec
Ended Job = job_1559386504130_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.11 sec   HDFS Read: 9416 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 110 msec
OK
59367
Time taken: 31.512 seconds, Fetched: 1 row(s)
hive>
```

Below screenshot shows some records from lookup_data_hbase table

```
hive> select * from lookup_data_hbase limit 20;
OK
340028465709212 1.6331555548882348E7    233     35775   2018-11-22 11:08:25
340054675199675 1.4156079786189131E7    631     54499   2018-08-30 00:24:04
340082915339645 1.5285685330791473E7    407     75772   2018-08-17 18:35:42
340134186926007 1.5239767522438556E7    614     65055   2018-10-30 11:24:46
340265728490548 1.608491671255562E7     202     72411   2018-05-28 12:51:31
340268219434811 1.2507323937605347E7    415     73041   2018-07-29 11:49:54
340379737226464 1.4198310998368107E7    229     18054   2018-05-30 15:41:44
340383645652108 1.4091750460468251E7    645     98027   2018-10-29 14:28:48
340803866934451 1.0843341196185412E7    502     64642   2018-04-24 09:29:27
340889618969736 1.3217942365515321E7    330     28009   2018-06-29 17:07:40
340924125838453 1.1712496575647112E7    644     45880   2018-08-31 09:17:44
341005627432127 1.2843150457220368E7    629     70710   2018-09-27 13:56:56
341029651579925 1.384435212566918E7     682     84750   2018-11-28 01:42:28
341311317050937 1.2104977216410311E7    440     31066   2018-02-26 01:54:36
341344252914274 1.0770448614621945E7    643     19432   2018-05-27 22:39:16
341363858179050 1.2514602030066479E7    559     15685   2018-09-28 04:09:12
341519629171378 1.529211229141615E7     408     72377   2018-10-31 10:35:18
341641153427489 1.3421215715089727E7    475     64682   2018-09-25 20:05:00
341719092861087 1.1521372646859605E7    442     36740   2018-11-18 21:54:51
341722035429601 1.2890883282580867E7    552     15488   2018-06-23 16:05:37
Time taken: 0.096 seconds, Fetched: 20 row(s)
hive>
```

Below screenshot shows count of records in lookup_data_hbase table

```
hive> select count(*) from lookup_data_hbase;
Query ID = ec2-user_20190601114848_1e534600-e31e-4b0f-a8d2-2354bc4cd7e8
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_1559386504130_0004, Tracking URL = http://ip-172-31-91-95.ec2.internal:8088/proxy/application_1559386504130_0004/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1559386504130_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-06-01 11:48:35,683 Stage-1 map = 0%,  reduce = 0%
2019-06-01 11:48:43,038 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.09 sec
2019-06-01 11:48:49,269 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.95 sec
MapReduce Total cumulative CPU time: 6 seconds 950 msec
Ended Job = job_1559386504130_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.95 sec   HDFS Read: 8799 HDFS Write: 4 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 950 msec
OK
999
Time taken: 29.962 seconds, Fetched: 1 row(s)
hive>
```

Below screenshot shows count of records from card_transactions_hbase where status is GENUINE

```
hive> select count(*) from card_transactions_hbase where status = 'GENUINE';
Query ID = ec2-user_20190601115151_fc8a73d1-b604-43fc-8ece-1f7a20c4d441
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_1559386504130_0005, Tracking URL = http://ip-172-31-91-95.ec2.internal:8088/proxy/application_1559386504130_0005/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1559386504130_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-06-01 11:51:19,562 Stage-1 map = 0%,  reduce = 0%
2019-06-01 11:51:27,845 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.2 sec
2019-06-01 11:51:35,176 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.26 sec
MapReduce Total cumulative CPU time: 10 seconds 260 msec
Ended Job = job_1559386504130_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.26 sec   HDFS Read: 10467 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 260 msec
OK
59150
Time taken: 32.119 seconds, Fetched: 1 row(s)
hive>
```

Below screenshot shows count of records from card_transactions_hbase table where status is FRAUD

```
hive> select count(*) from card_transactions_hbase where status = 'FRAUD';
Query ID = ec2-user_20190601115252_909d487e-d445-4328-8bea-6d50cbc2e33c
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_1559386504130_0006, Tracking URL = http://ip-172-31-91-95.ec2.internal:8088/proxy/application_1559386504130_0006/
Kill Command = /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop/bin/hadoop job  -kill job_1559386504130_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-06-01 11:52:31,317 Stage-1 map = 0%,  reduce = 0%
2019-06-01 11:52:39,627 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 6.71 sec
2019-06-01 11:52:45,818 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.8 sec
MapReduce Total cumulative CPU time: 9 seconds 800 msec
Ended Job = job_1559386504130_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.8 sec   HDFS Read: 10514 HDFS Write: 4 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 800 msec
OK
217
Time taken: 33.419 seconds, Fetched: 1 row(s)
hive>
```

Showing processing of transactions for a card id – 6599900931314251. This is the same card id for which I had shown screenshots in mid submission as well. In continuation, I picked up the same card id.

Below are the transaction details for this card id received from Kafka topic captured in Output.txt. There are 7 transactions to be processed for this card id.


```
{"card_id":6599900931314251,"member_id":928036864799687,"amount":1447809,"postcode":24848,"pos_id":302116371961141,"transaction_dt":"02-10-2018 12:49:01"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":9723817,"postcode":72043,"pos_id":239218493122711,"transaction_dt":"08-10-2018 10:06:47"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":6902536,"postcode":99739,"pos_id":97601123545965,"transaction_dt":"12-02-2018 12:56:07"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":5471142,"postcode":61237,"pos_id":430322863074336,"transaction_dt":"09-03-2018 20:30:08"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":4361135,"postcode":97026,"pos_id":346155771351832,"transaction_dt":"12-10-2018 03:49:30"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":4876716,"postcode":49910,"pos_id":683867473927303,"transaction_dt":"21-11-2018 05:59:02"}
{"card_id":6599900931314251,"member_id":928036864799687,"amount":2179073,"postcode":34270,"pos_id":229855333287383,"transaction_dt":"27-06-2018 14:17:28"}
```

Below are the screenshots of processing of transactions for this card id captured in Output.txt

Transaction 1:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 1447809.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 97423

Current Post Code for Card ID of current transaction is : 24848

Last Transaction Date for Card ID of current transaction is : 2018-01-31 11:25:16

Current Transaction Date for Card ID of current transaction is : 2018-10-02 12:49:01

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 3644.2925949321507

Absolute Time Difference between current and last transaction (in seconds) is : 21086625

Speed (Distance / Time) : 1.7282484015019713E-4 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 1447809.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 1.7282484015019713E-4 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:42

================================================================
```

Transaction 2:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 9723817.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 24848

Current Post Code for Card ID of current transaction is : 72043

Last Transaction Date for Card ID of current transaction is : 2018-10-02 12:49:01

Current Transaction Date for Card ID of current transaction is : 2018-10-08 10:06:47

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 892.4283020933965

Absolute Time Difference between current and last transaction (in seconds) is : 508666

Speed (Distance / Time) : 0.0017544485027373492 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 9723817.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 0.0017544485027373492 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:42

================================================================
```

Transaction 3:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 6902536.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 72043

Current Post Code for Card ID of current transaction is : 99739

Last Transaction Date for Card ID of current transaction is : 2018-10-08 10:06:47

Current Transaction Date for Card ID of current transaction is : 2018-02-12 12:56:07

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 5598.51251087252

Absolute Time Difference between current and last transaction (in seconds) is : 20553040

Speed (Distance / Time) : 2.723934031594606E-4 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 6902536.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 2.723934031594606E-4 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:42

================================================================
```

Transaction 4:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 5471142.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 99739

Current Post Code for Card ID of current transaction is : 61237

Last Transaction Date for Card ID of current transaction is : 2018-02-12 12:56:07

Current Transaction Date for Card ID of current transaction is : 2018-03-09 20:30:08

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 5082.797030871786

Absolute Time Difference between current and last transaction (in seconds) is : 2187241

Speed (Distance / Time) : 0.002323839499566708 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 5471142.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 0.002323839499566708 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:42

================================================================
```

Transaction 5:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 4361135.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 61237

Current Post Code for Card ID of current transaction is : 97026

Last Transaction Date for Card ID of current transaction is : 2018-03-09 20:30:08

Current Transaction Date for Card ID of current transaction is : 2018-10-12 03:49:30

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 2616.258781260241

Absolute Time Difference between current and last transaction (in seconds) is : 18688762

Speed (Distance / Time) : 1.3999101605875451E-4 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 4361135.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 1.3999101605875451E-4 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:42

================================================================
```

Transaction 6:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:42

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 4876716.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 97026

Current Post Code for Card ID of current transaction is : 49910

Last Transaction Date for Card ID of current transaction is : 2018-10-12 03:49:30

Current Transaction Date for Card ID of current transaction is : 2018-11-21 05:59:02

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 2567.88379389069

Absolute Time Difference between current and last transaction (in seconds) is : 3463772

Speed (Distance / Time) : 7.41354740984883E-4 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 4876716.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 7.41354740984883E-4 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:43

================================================================
```

Transaction 7:

```
================================================================

New Transaction Processing Start Time : 2019-06-01 11:15:43

======== Classify Current Transaction into GENUINE/FRAUD ==========

Card ID of current transaction is : 6599900931314251

UCL (Upper Control Limit) for Card ID of current transaction is : 1.2121408572464656E7

Current Transaction Amount is : 2179073.0

Credit score for Card ID of current transaction is : 297

Last Post Code for Card ID of current transaction is : 49910

Current Post Code for Card ID of current transaction is : 34270

Last Transaction Date for Card ID of current transaction is : 2018-11-21 05:59:02

Current Transaction Date for Card ID of current transaction is : 2018-06-27 14:17:28

Distance (returned by DistanceUtility) between current and last postcode (in km) is : 2222.340348620956

Absolute Time Difference between current and last transaction (in seconds) is : 12670894

Speed (Distance / Time) : 1.7538938835893946E-4 km/second

Current Card Transaction has passed Credit Score Validation rule : Score : 297 >= 200

Current Card Transaction has passed UCL (Upper Control Limit) Validation rule : Amount : 2179073.0 <= 1.2121408572464656E7

Current Card Transaction has passed Zip Code Distance Validation rule : Speed (km/sec) : 1.7538938835893946E-4 <= 0.25

Current Card Transaction has passed all 3 Validation rules and so status is : GENUINE

========== Update NoSQL Database for Current Transaction ==========

Card ID of current transaction is : 6599900931314251

Current Transaction Details are populated in Card Transactions HBase Table : card_transactions_hive

Current Card Transaction status is GENUINE so updating lookup table

Postcode and Transaction Date updated for Card ID of current transaction in lookup HBase table : lookup_data_hive

================================================================

Current Transaction Processing End Time : 2019-06-01 11:15:43

================================================================
```

Below are screenshots from HBase for this card id:

First screenshot shows postcode and transaction_dt for same card id in lookup table

You will notice 10 VERSIONS for postcode and transaction_dt. This column family is set to store up to 10 VERSIONS so all 10 VERSIONS are displayed. 3 versions came at the time of mid submission because I had populated lookup table 3 times. First, when I populated lookup table for first time in task 3. Second, when I populated lookup table for second time in task 4 while running oozie job without coordinator. Third, when I populated lookup table for third time in task 4 again while running oozie job with coordinator this time. Rest 7 versions came from processing of transactions for this card id. There were 7 transactions fetched for Kafka topic for this card and all 7 were GENUINE as shown in screenshots above already.

```
hbase(main):003:0> get 'lookup_data_hive', '6599900931314251', {COLUMN => ['lookup_transaction_family:postcode', 'lookup_transaction_family:transaction_dt'], VERSIONS=>10}
COLUMN                                        CELL
 lookup_transaction_family:postcode           timestamp=1559387743032, value=34270
 lookup_transaction_family:postcode           timestamp=1559387743012, value=49910
 lookup_transaction_family:postcode           timestamp=1559387742983, value=97026
 lookup_transaction_family:postcode           timestamp=1559387742962, value=61237
 lookup_transaction_family:postcode           timestamp=1559387742941, value=99739
 lookup_transaction_family:postcode           timestamp=1559387742920, value=72043
 lookup_transaction_family:postcode           timestamp=1559387742899, value=24848
 lookup_transaction_family:postcode           timestamp=1558807804067, value=97423
 lookup_transaction_family:postcode           timestamp=1558774131995, value=97423
 lookup_transaction_family:postcode           timestamp=1558772840191, value=97423
 lookup_transaction_family:transaction_dt     timestamp=1559387743032, value=2018-06-27 14:17:28
 lookup_transaction_family:transaction_dt     timestamp=1559387743012, value=2018-11-21 05:59:02
 lookup_transaction_family:transaction_dt     timestamp=1559387742983, value=2018-10-12 03:49:30
 lookup_transaction_family:transaction_dt     timestamp=1559387742962, value=2018-03-09 20:30:08
 lookup_transaction_family:transaction_dt     timestamp=1559387742941, value=2018-02-12 12:56:07
 lookup_transaction_family:transaction_dt     timestamp=1559387742920, value=2018-10-08 10:06:47
 lookup_transaction_family:transaction_dt     timestamp=1559387742899, value=2018-10-02 12:49:01
 lookup_transaction_family:transaction_dt     timestamp=1558807804067, value=2018-01-31 11:25:16
 lookup_transaction_family:transaction_dt     timestamp=1558774131995, value=2018-01-31 11:25:16
 lookup_transaction_family:transaction_dt     timestamp=1558772840191, value=2018-01-31 11:25:16
20 row(s) in 0.0190 seconds

hbase(main):004:0>
```

Below screenshot shows ucl and score for same card id in lookup table. This column family is set to store up to 1 VERSION only so only 1 VERSION is displayed.

```
hbase(main):004:0> get 'lookup_data_hive', '6599900931314251', {COLUMN => ['lookup_card_family:ucl', 'lookup_card_family:score'], VERSIONS=>10}
COLUMN                                        CELL
 lookup_card_family:score                     timestamp=1558807804067, value=297
 lookup_card_family:ucl                       timestamp=1558807804067, value=1.2121408572464656E7
2 row(s) in 0.0080 seconds

hbase(main):005:0>
```

Below screenshot shows latest version of all columns for this card id in lookup table

```
hbase(main):005:0> get 'lookup_data_hive', '6599900931314251'
COLUMN                                        CELL
 lookup_card_family:score                     timestamp=1558807804067, value=297
 lookup_card_family:ucl                       timestamp=1558807804067, value=1.2121408572464656E7
 lookup_transaction_family:postcode           timestamp=1559387743032, value=34270
 lookup_transaction_family:transaction_dt     timestamp=1559387743032, value=2018-06-27 14:17:28
4 row(s) in 0.0070 seconds

hbase(main):006:0>
```

========== Output – END ==========

========== Final Submission – END ==========