STREAMING DATA & DETECTION OF FRAUD

For streaming the data following python files are required :

- Driver.py
- Dao.py
- Rules.py
- geo_map.py

GEO_MAP.PY

This file is used to initialize Class GEO_Map which is used to calculate distance between the current location where the transaction of a given card happens and the location of the last transaction of this card.

For each location, it will have different post code, based on this post code, we can get its latitude and longitude. Therefore, we will use this information to calculate the distance between the current and last transaction of a given card.

RULES.PY

We will develop verify functions in this file to check whether a transaction is FRAUD or not.

1. The first rule is score > 200

```
def verify_score(card_id):
    try:
        conn = MongoClient();
        db = conn["transaction_db"]
        lookupTable = db["lookup_table"]
        card_info = lookupTable.find_one({'card_id': card_id})

        if int(card_info["score"]) > 200:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

The input parameter of this function is card_id, it will get score of this card in the look-up table then check if the score is greater than 200 or not. If its score greater than 200, it returns True, the first rule is passed.

2. The second rule is amount < UCL value

```
def verify_ucl(card_id, amount):
    try:
        conn = MongoClient();
        db = conn["transaction_db"]
        lookupTable = db["lookup_table"]
        card_info = lookupTable.find_one({'card_id': card_id})

        if amount < float(card_info["ucl"]):
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

The input parameters of this function are card_id and amount of this transaction. It will base on the card_id, then access look-up table to get UCL value of this card, then compare the UCL value with the amount parameter. If the amount < UCL value, it returns True, the second rule is passed.

3. The third rule is for checking post code of the transaction.

The idea of this rule is check location and execution time of the current and last transaction of a given card. Based on these information, it can calculate the distance and the elapsed time between the 2 transactions, then it can calculate the velocity of the owner of this card is higher than the speed limit or not. If the velocity is too high and over the speed limit, the transaction is considered FRAUD.

First, we need one more function to calculate the speed/velocity. This function will receive distance between the 2 transactions and their execution time.

```
def speed_calc(dist, trans_dt, last_trans_dt):
    trans_dt      = datetime.strptime(trans_dt,      '%d-%m-%Y %H:%M:%S')
    last_trans_dt = datetime.strptime(last_trans_dt, '%d-%m-%Y %H:%M:%S')
    consumed_time = (trans_dt - last_trans_dt).total_seconds()

    try:
        return dist / consumed_time
    except ZeroDivisionError:
        return 299792.458
```

Then it will take the distance divide to the consumed time (the time between the 2 transactions) to return the velocity.

In the verify post code function, we will get card_id, its current post code and transaction time.

Based on the card ID, we will access the lookup table to get post code and transaction time of the last transaction of this card. Then we will call the class geo_map to take latitude and longitude of the post code of the last transaction. We also perform that to take latitude and longitude of the current transaction. Then call function distance from geo_map to calculate distance between locations of the 2 transactions.

After getting the distance, we will use it for the function speed_calc to calculatethe velocity.

Finally, compare the velocity with the speed limit. The speed limit is converted to km/s, 900 km/h = 0.25 km/s. If the velocity is greater than speed limit, the transaction is considered FRAUD. Otherwise, the third rule is passed.

```
def verify_postcode(card_id, postcode, trans_dt):
    try:
        geo_map = GEO_Map.get_instance()
        conn = MongoClient();
        db = conn["transaction_db"]
        lookupTable = db["lookup_table"]
        card_info = lookupTable.find_one({'card_id': card_id})
        last_postcode = card_info['postcode']
        last_trans_dt = card_info['transaction_dt']

        current_lat = geo_map.get_lat(str(postcode))
        for data in current_lat:
            current_lat1 = data
        current_long = geo_map.get_long(str(postcode))
        for data in current_long:
            current_long1 = data
        previous_lat = geo_map.get_lat(str(last_postcode))
        for data in previous_lat:
            previous_lat1 = data
        previous_long = geo_map.get_long(str(last_postcode))
        for data in previous_long:
            previous_long1 = data

        distance = geo_map.distance(lat1=current_lat1, long1=current_long1, lat2=previous_lat1,
long2=previous_long1)

        veloc = speed_calc(distance, trans_dt, last_trans_dt)

        if veloc < speed_limit:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

Finally, we prepare a summary functions to check if the transaction is passed all3 rules.

```
def verify_rules(card_id, amount, postcode, transaction_dt):
    rule_ucl      = verify_ucl(card_id, amount)
    rule_score    = verify_score(card_id)
    rule_postcode = verify_postcode(card_id, postcode, transaction_dt)

    if (rule_ucl==True and rule_score==True and rule_postcode==True):
        return True
    else :
        return False
```

If a given transaction passes all 3 rules, it will be GENUINE.

DAO.PY

This file includes all operation applied to MongoDB database.

1. Function insert into table card_transactions

We will call this function whenever Kafka Consumer receives a new transaction.

```
def insertCardTrans(trans, Status):
    conn = MongoClient();
    db = conn["transaction_db"]
    coll = db["card_transactions"]

    if (Status==True):
        val_Status = "GENUINE"
    else:
        val_Status = "FRAUD"

    doc = {"card_id" : trans["card_id"],
           "member_id" : trans["member_id"],
           "amount" : trans["amount"],
           "postcode" : trans["postcode"],
           "pos_id" : trans["pos_id"],
           "transaction_dt" : trans["transaction_dt"],
           "status" : val_Status}
    coll.insert_one(doc)
```

2. Function update data in table tb_lookup

This function is used to update post code and transaction time of a given cardin lookup table when the transaction of this card is considered GENUINE.

```
def updateLookup(trans):
    conn = MongoClient();
    db = conn["transaction_db"]
    coll = db["lookup_table"]

    condition = {"card_id" : trans["card_id"]}
    newDoc = {"$set": {"postcode" : trans["postcode"],
                       "transaction_dt" : trans["transaction_dt"]}}
    coll.update_one(condition, newDoc)
```

DRIVER.PY

It is the main file of this project. We will execute this file to receive streaming data from Kafka and call processing functions from other files to check each streaming transactions, then perform action to database.

```
from kafka import KafkaConsumer
from rules import *
from dao import *
import sys
import datetime
import json

bootstrap_servers = ['18.211.252.152:9092']
topicName = 'transactions-topic-verified'
consumer = KafkaConsumer (topicName, bootstrap_servers = bootstrap_servers, auto_offset_reset =
'earliest')

try:
    for message in consumer:
        trans = json.loads(message.value)
        status = verify_rules(trans["card_id"], trans["amount"], trans["postcode"], trans["transaction_dt"])
        insertCardTrans(trans, status)
        print("Transaction has been inserted into card transactions table.")
        if (status==True):
            updateLookup(trans)
            print("Last successful transaction has been updated!")
        else:
            print('There is suspicious transaction! FRAUD!')

except KeyboardInterrupt:
    sys.exit()
```

Each streaming transaction is sent by Kafka under JSON format, then it will be processed by verify_rules function.

All transactions will be inserted into card_transactions table (collection) in MongoDB. And only GENUINE transactions are updated in look-up table.

ORGANIZE AND EXECUTE PROJECT

It is due to this is a quite simple project, so I store all python file in only one folder, just focus on processing streaming data. To run this project, we just run the python file driver.py

Here is the result

```
Transaction has been inserted into card transactions table.
There is suspicious transaction! FRAUD!
Transaction has been inserted into card transactions table.
Last successful transaction has been updated!
Transaction has been inserted into card transactions table.
Last successful transaction has been updated!
Transaction has been inserted into card transactions table.
Last successful transaction has been updated!
Transaction has been inserted into card transactions table.
Last successful transaction has been updated!
Transaction has been inserted into card transactions table.
Last successful transaction has been updated!
Transaction has been inserted into card transactions table.
```

We can also check the number of row in card_transactions collection inMongoDB to see the new documents are inserted.

```
transaction_db> db.card_transactions.count()
55393
transaction_db> db.card_transactions.count()
56117
transaction_db> db.card_transactions.count()
57507
```

We can see the data in card_transactions collection is inserted continuously.