Final Submission

Real-Time Fraud Detection System Using Spark Streaming ,Kafka and HBase

Introduction

This project involves processing streaming data from Kafka, applying business rules to detect fraudulent transactions in real-time, and writing the results to an HBase database. The project leverages Spark Streaming for real-time processing, HBase for data storage and retrieval, and custom user-defined functions (UDFs) for implementing business logic.

Problem Approach

1. **Reading Source Card Transactions from Kafka:**
    - Stream card transaction data from a Kafka topic named `transactions-topic-verified`.
    - Define a schema to structure the streaming DataFrame according to the expected data format.
2. **Deriving the "status" Column:**
    - The status column is determined with two possible values: "GENUINE" or "FRAUD". This is based on the evaluation of three rules:
3. **a. Credit Score:**
    - Retrieve the credit score from the HBase card lookup table using the card ID from the transaction data.
    - If the score is less than 200, label the transaction as "FRAUD" and move to the next transaction. Otherwise, proceed to the next rule.
4. **b. Upper Control Limit (UCL):**
    - Compare the transaction amount with the UCL value from the HBase card lookup table.
    - If the amount exceeds the UCL, label the transaction as "FRAUD" and move to the next transaction. Otherwise, proceed to the final rule.
5. **c. Zip Code Distance:**
    - Calculate the distance between the locations of the two most recent transactions and measure the time difference.
    - Compute the speed in kilometers per second and compare it against a threshold of 0.25 kilometers per second (900 kilometers per hour, the average speed of commercial flights).

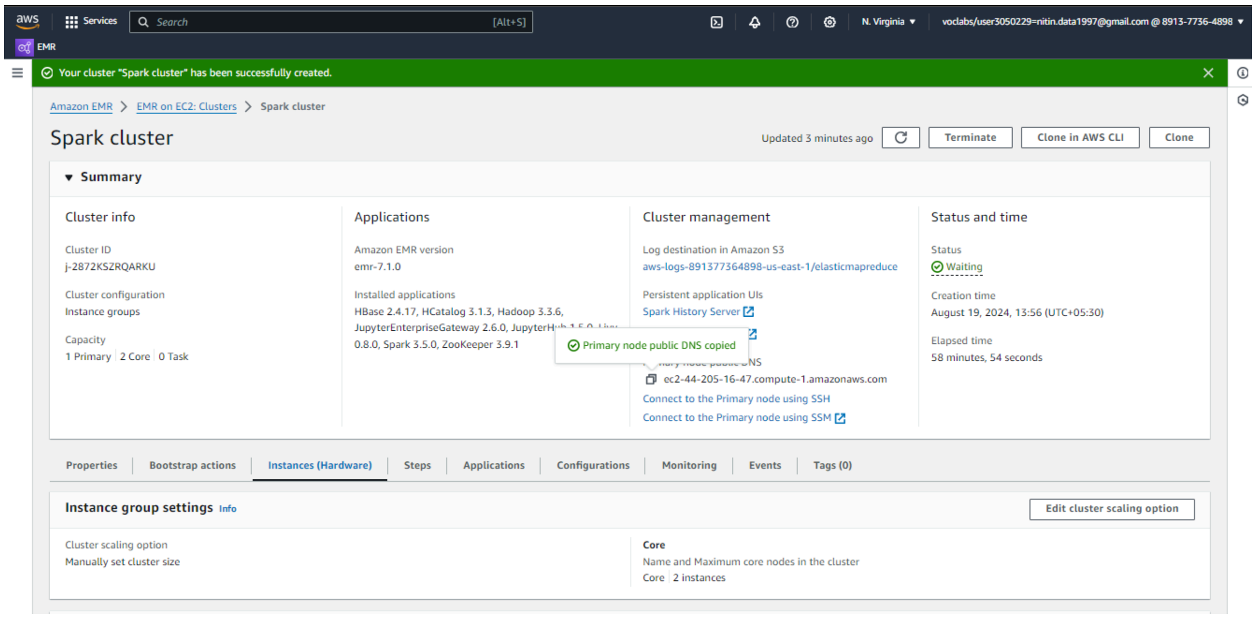
    - If the speed exceeds this threshold, label the transaction as "FRAUD". Otherwise, label it as "GENUINE".
6. **Inserting Processed Data into HBase:**
    - Insert the processed card transaction data, along with its status, into the `card_transactions` table in HBase.
7. **Updating Postcode and Transaction Date in HBase:**
    - Update the postcode and transaction date in the card lookup table in HBase, but only for transactions labeled as "GENUINE".
8. **Displaying Processed Transactions in the Console:**
    - After processing all transactions in the current micro-batch, display the transaction details along with their status in the console for real-time analysis.

---

1. Cluster Configuration:

We have created 3 Node cluster with m4.xlarge configuration.

1 Primary 2 Core	M4.xlarge M5.xlarge



Your cluster "Spark cluster" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > Spark cluster

Spark cluster

Summary

Cluster info

Cluster ID
j-2872KSZRQARKU

Cluster configuration
Instance groups

Capacity
1 Primary | 2 Core | 0 Task

Applications

Amazon EMR version
emr-7.1.0

Installed applications
HBase 2.4.17, HCatalog 3.1.3, Hadoop 3.3.6, JupyterEnterpriseGateway 2.6.0, JupyterH
0.8.0, Spark 3.5.0, ZooKeeper 3.9.1

Cluster management

Log destination in Amazon S3
aws-logs-891377364898-us-east-1/elasticmapreduce

Persistent application UIs
Spark History Server

Primary node public DNS copied

NS
ec2-44-205-16-47.compute-1.amazonaws.com
Connect to the Primary node using SSH
Connect to the Primary node using SSM

Status and time

Status
Waiting

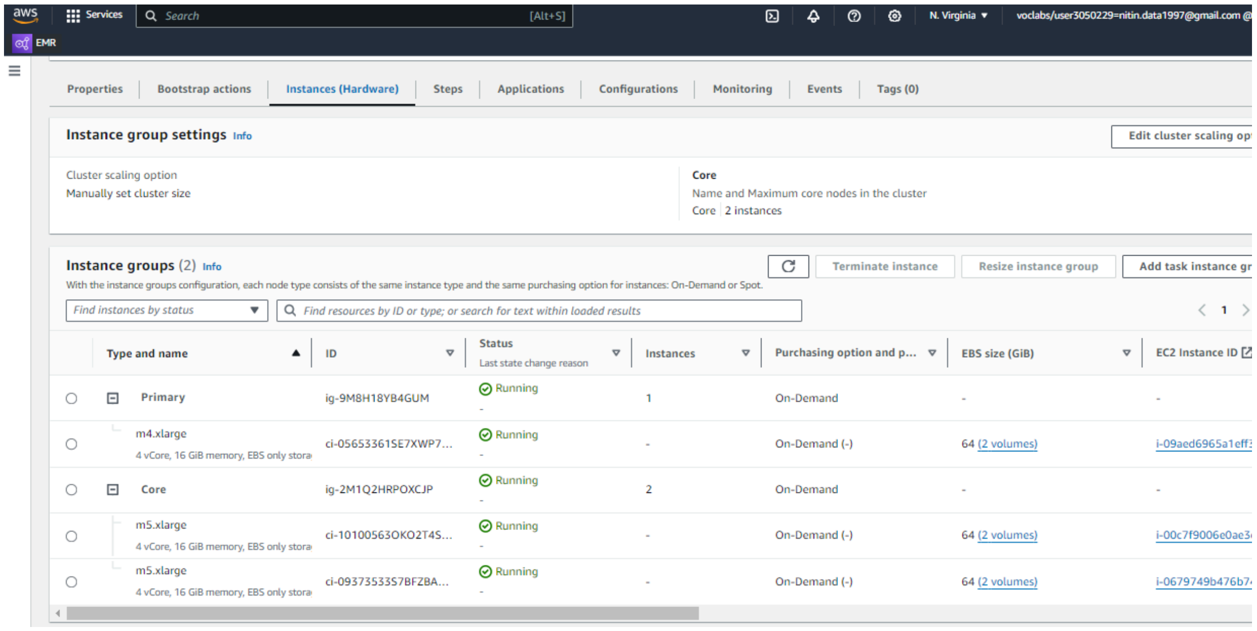
Creation time
August 19, 2024, 13:56 (UTC+05:30)

Elapsed time
58 minutes, 54 seconds

Instance group settings

Cluster scaling option
Manually set cluster size

Core
Name and Maximum core nodes in the cluster
Core | 2 instances



Instance group settings Info

Edit cluster scaling op

Cluster scaling option
Manually set cluster size

Core
Name and Maximum core nodes in the cluster
Core | 2 instances

Instance groups (2) Info

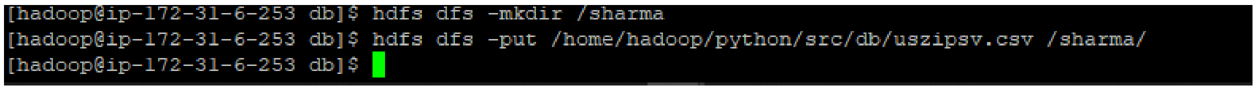
Terminate instance | Resize instance group | Add task instance gr

With the instance groups configuration, each node type consists of the same instance type and the same purchasing option for instances: On-Demand or Spot.

Find instances by status | Find resources by ID or type; or search for text within loaded results

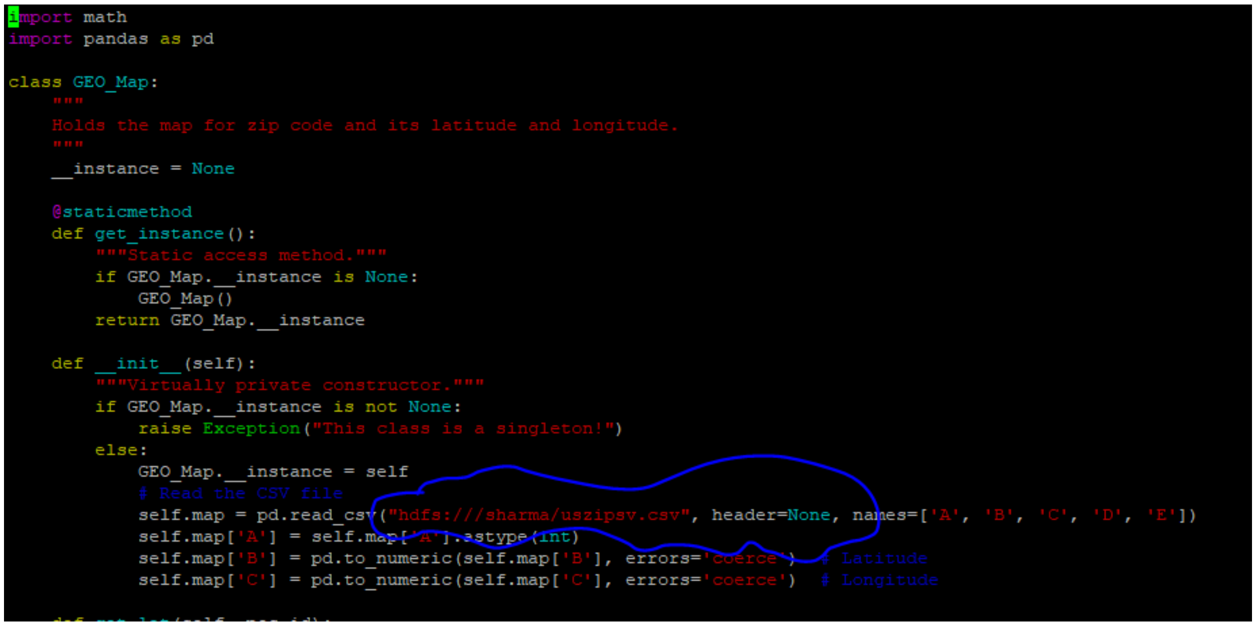
Type and name	ID	Status Last state change reason	Instances	Purchasing option and p...	EBS size (GiB)	EC2 Instance ID
Primary	ig-9M8H18YB4GUM	Running -	1	On-Demand	-	-
m4.xlarge 4 vCore, 16 GiB memory, EBS only stora	ci-05653361SE7XWP7...	Running -	-	On-Demand (-)	64 (2 volumes)	i-09aed6965a1eff
Core	ig-2M1Q2HRPOXCJP	Running -	2	On-Demand	-	-
m5.xlarge 4 vCore, 16 GiB memory, EBS only stora	ci-10100563OKO2T4S...	Running -	-	On-Demand (-)	64 (2 volumes)	i-00c7f9006e0ae3
m5.xlarge 4 vCore, 16 GiB memory, EBS only stora	ci-09373533S7BFZBA...	Running -	-	On-Demand (-)	64 (2 volumes)	i-0679749b476b7

Note: Add uszipsv.csv in hdfs dfs -put /home/hadoop/python/src/db/uszipsv.csv /sharma/



```
[hadoop@ip-172-31-6-253 db]$ hdfs dfs -mkdir /sharma
[hadoop@ip-172-31-6-253 db]$ hdfs dfs -put /home/hadoop/python/src/db/uszipsv.csv /sharma/
[hadoop@ip-172-31-6-253 db]$
```

and geo_map.py access file through this directory as this file need to be accessed by core node as well primary_node



```
import math
import pandas as pd

class GEO_Map:
    """
    Holds the map for zip code and its latitude and longitude.
    """
    __instance = None

    @staticmethod
    def get_instance():
        """Static access method."""
        if GEO_Map.__instance is None:
            GEO_Map()
        return GEO_Map.__instance

    def __init__(self):
        """Virtually private constructor."""
        if GEO_Map.__instance is not None:
            raise Exception("This class is a singleton!")
        else:
            GEO_Map.__instance = self
            # Read the CSV file
            self.map = pd.read_csv("hdfs:///sharma/uszipsv.csv", header=None, names=['A', 'B', 'C', 'D', 'E'])
            self.map['A'] = self.map['A'].astype(int)
            self.map['B'] = pd.to_numeric(self.map['B'], errors='coerce')  # Latitude
            self.map['C'] = pd.to_numeric(self.map['C'], errors='coerce')  # Longitude
```

Need to install the below codes on all the node.

```
sudo su
sudo yum groupinstall 'Development Tools' -y
sudo yum install python3-devel -y
pip3 install happybase
pip3 install pandas
pip install fsspec
pip install pyarrow
```

`sudo su`: Switch to the root user to gain administrative privileges.

`sudo yum groupinstall 'Development Tools' -y`: Install a group of essential development tools (like compilers and libraries) on a Linux system.

`sudo yum install python3-devel -y`: Install the Python 3 development libraries required for compiling Python modules.

`pip3 install happybase`: Install the `happybase` Python library, which is used to interact with HBase.

`pip3 install pandas`: Install the `pandas` library, commonly used for data manipulation and analysis.

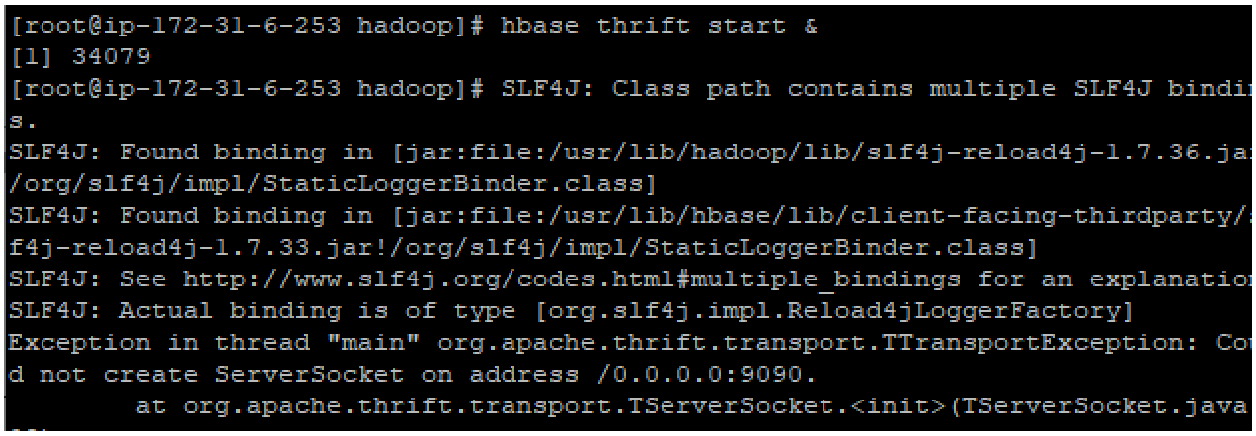
`pip install fsspec`: Install the `fsspec` library, which provides a unified interface to various file systems.

`pip install pyarrow`: Install the `pyarrow` library, which is used for efficient data interchange between Python and other systems, often in the context of Apache Arrow.

THRIFT SERVER

hbase thrift start &

Run above command to start thrift server so that hbase can interact with code as well nodes

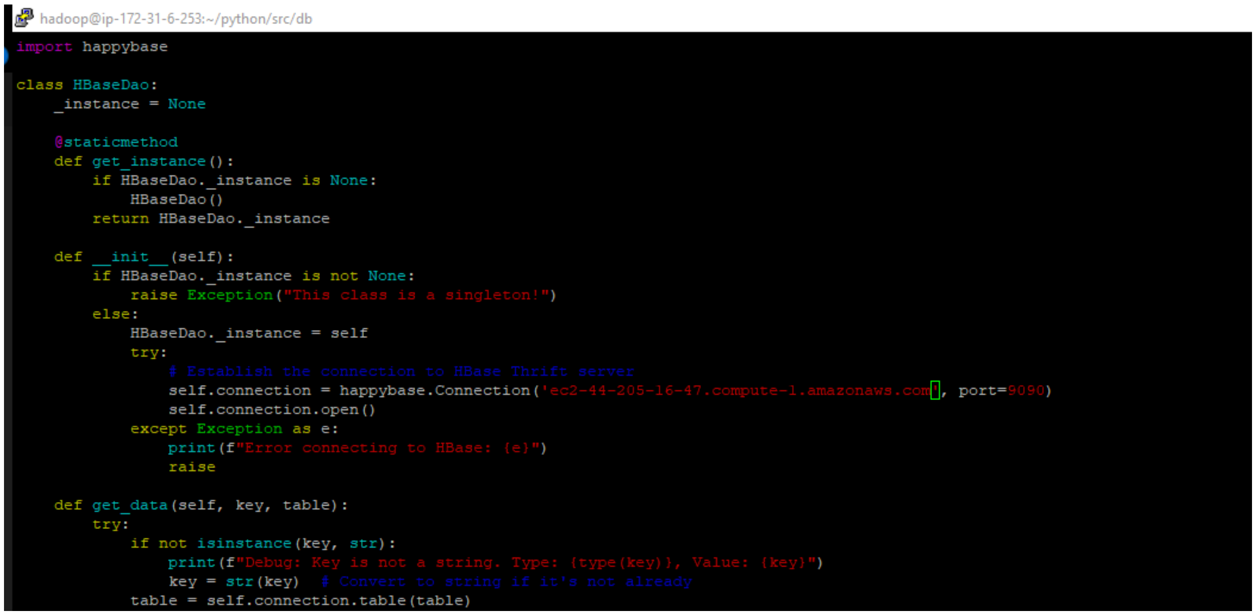


```
[root@ip-172-31-6-253 hadoop]# hbase thrift start &
[1] 34079
[root@ip-172-31-6-253 hadoop]# SLF4J: Class path contains multiple SLF4J bindi
s.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-reload4j-1.7.36.ja
/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hbase/lib/client-facing-thirdparty/
f4j-reload4j-1.7.33.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanatio
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
Exception in thread "main" org.apache.thrift.transport.TTransportException: Co
d not create ServerSocket on address /0.0.0.0:9090.
        at org.apache.thrift.transport.TServerSocket.<init>(TServerSocket.java
```

---

dao.py

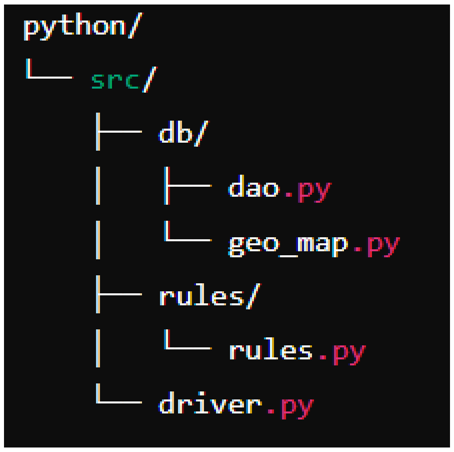
Update this file with your master node DNS



hadoop@ip-172-31-6-253:~/python/src/db

```
import happybase

class HBaseDao:
    _instance = None

    @staticmethod
    def get_instance():
        if HBaseDao._instance is None:
            HBaseDao()
        return HBaseDao._instance

    def __init__(self):
        if HBaseDao._instance is not None:
            raise Exception("This class is a singleton!")
        else:
            HBaseDao._instance = self
            try:
                # Establish the connection to HBase Thrift server
                self.connection = happybase.Connection('ec2-44-205-16-47.compute-1.amazonaws.com', port=9090)
                self.connection.open()
            except Exception as e:
                print(f"Error connecting to HBase: {e}")
                raise

    def get_data(self, key, table):
        try:
            if not isinstance(key, str):
                print(f"Debug: Key is not a string. Type: {type(key)}, Value: {key}")
                key = str(key)  # Convert to string if it's not already
            table = self.connection.table(table)
```

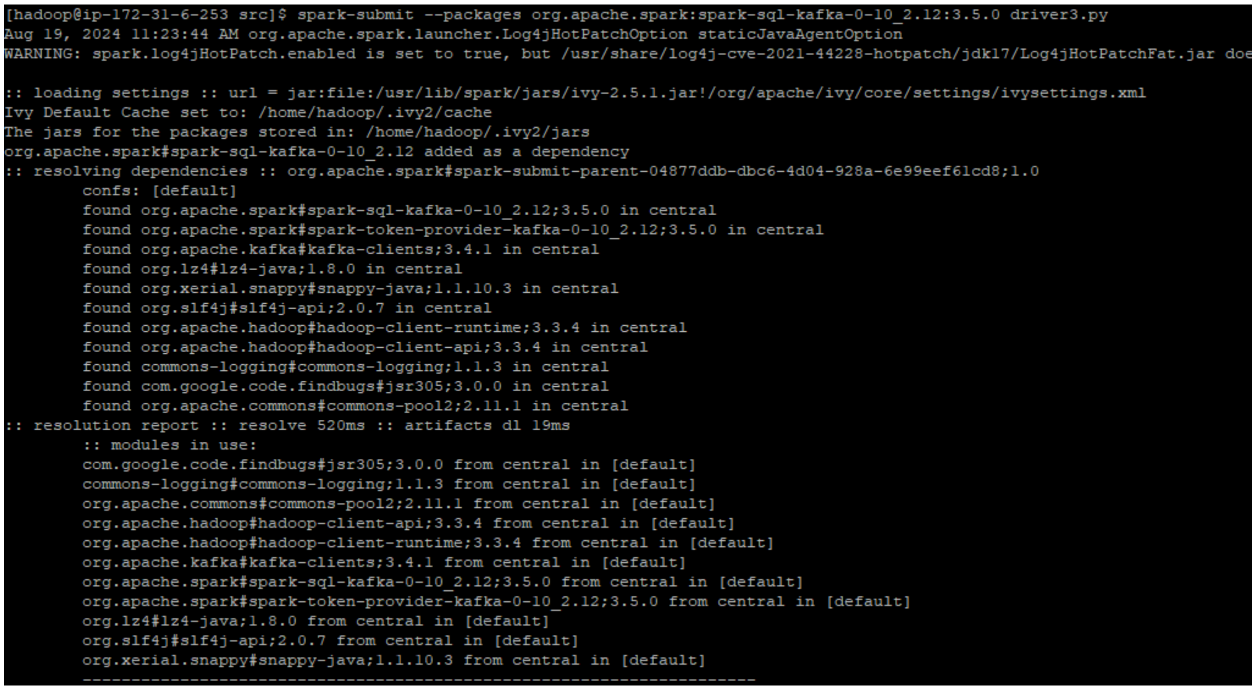
SPARK-SUBMIT



```
python/
└── src/
    ├── db/
    │   ├── dao.py
    │   └── geo_map.py
    ├── rules/
    │   └── rules.py
    └── driver.py
```

To execute the driver PySpark program, follow the provided commands

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 driver.py
```



```
[hadoop@ip-172-31-6-253 src]$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 driver3.py
Aug 19, 2024 11:23:44 AM org.apache.spark.launcher.Log4jHotPatchOption staticJavaAgentOption
WARNING: spark.log4jHotPatch.enabled is set to true, but /usr/share/log4j-cve-2021-44228-hotpatch/jdk17/Log4jHotPatchFat.jar doe

:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-04877ddb-dbc6-4d04-928a-6e99eef61cd8;1.0
        confs: [default]
        found org.apache.spark#spark-sql-kafka-0-10_2.12;3.5.0 in central
        found org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.5.0 in central
        found org.apache.kafka#kafka-clients;3.4.1 in central
        found org.lz4#lz4-java;1.8.0 in central
        found org.xerial.snappy#snappy-java;1.1.10.3 in central
        found org.slf4j#slf4j-api;2.0.7 in central
        found org.apache.hadoop#hadoop-client-runtime;3.3.4 in central
        found org.apache.hadoop#hadoop-client-api;3.3.4 in central
        found commons-logging#commons-logging;1.1.3 in central
        found com.google.code.findbugs#jsr305;3.0.0 in central
        found org.apache.commons#commons-pool2;2.11.1 in central
:: resolution report :: resolve 520ms :: artifacts dl 19ms
        :: modules in use:
        com.google.code.findbugs#jsr305;3.0.0 from central in [default]
        commons-logging#commons-logging;1.1.3 from central in [default]
        org.apache.commons#commons-pool2;2.11.1 from central in [default]
        org.apache.hadoop#hadoop-client-api;3.3.4 from central in [default]
        org.apache.hadoop#hadoop-client-runtime;3.3.4 from central in [default]
        org.apache.kafka#kafka-clients;3.4.1 from central in [default]
        org.apache.spark#spark-sql-kafka-0-10_2.12;3.5.0 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.5.0 from central in [default]
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.slf4j#slf4j-api;2.0.7 from central in [default]
        org.xerial.snappy#snappy-java;1.1.10.3 from central in [default]
        ---------------------------------------------------------------------
```

Driver file handles the following operations:

- a. Initializes the Spark session.
- b. Retrieves input data from Kafka.
- c. Defines the schema for the transaction data.
- d. Implements a User-Defined Function (UDF) that verifies transaction rules and updates the relevant lookup and master tables.
- e. Displays the processed data on the console.

---

1. Spark Configuration and Session Initialization

First, the necessary libraries are imported, and a Spark session is created with custom configurations.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark import SparkConf
from datetime import datetime
```

- **Spark Configuration:** Spark is configured to enable dynamic allocation of executors, with memory settings for both the executor and driver.

```
conf = SparkConf()
conf.set("spark.dynamicAllocation.enabled", "true")
conf.set("spark.dynamicAllocation.minExecutors", "2")
conf.set("spark.dynamicAllocation.maxExecutors", "5")
conf.set("spark.dynamicAllocation.initialExecutors", "2")
conf.set("spark.executor.memory", "4g")
conf.set("spark.driver.memory", "4g")
conf.set("spark.yarn.executor.memoryOverhead", "1024")
conf.set("spark.yarn.driver.memoryOverhead", "1024")
```

- **Spark Session:** A Spark session is then created, setting the application name and applying the above configuration.

```
spark = SparkSession.builder \
    .appName("CapStone_Project") \
    .config(conf=conf) \
    .getOrCreate()
```

- **Log Level:** The log level is set to `ERROR` to reduce verbosity in the console output.

```
spark.sparkContext.setLogLevel('ERROR')
```

- **Python File Addition:** Additional Python files needed for database operations, geographical data mapping, and fraud rules are added to the Spark context.

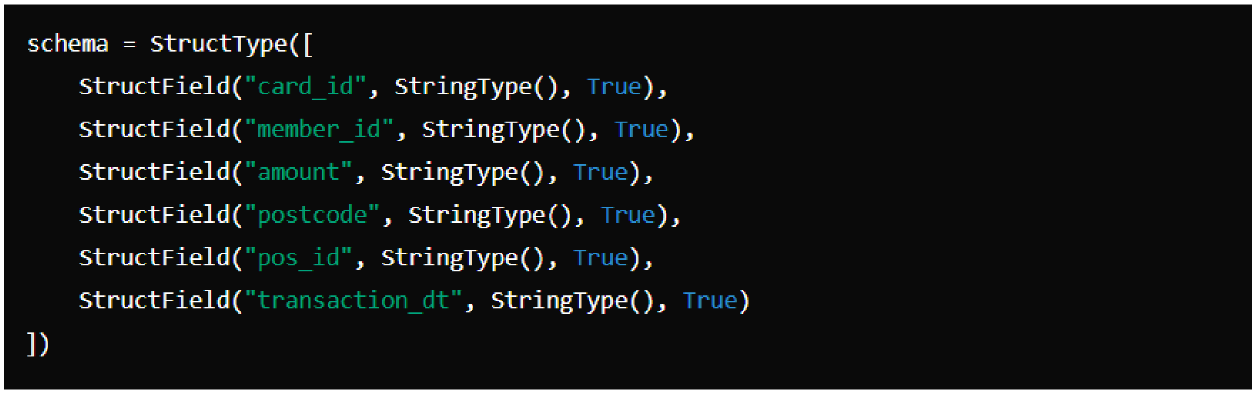
```
sc = spark.sparkContext
sc.addPyFile('/home/hadoop/python/src/db/dao.py')
sc.addPyFile('/home/hadoop/python/src/db/geo_map.py')
sc.addFile('/home/hadoop/python/src/rules/rules.py')
```

2. Reading and Parsing Streaming Data

- **Kafka Stream:** The streaming data is read from a Kafka topic named `transactions-topic-verified`.

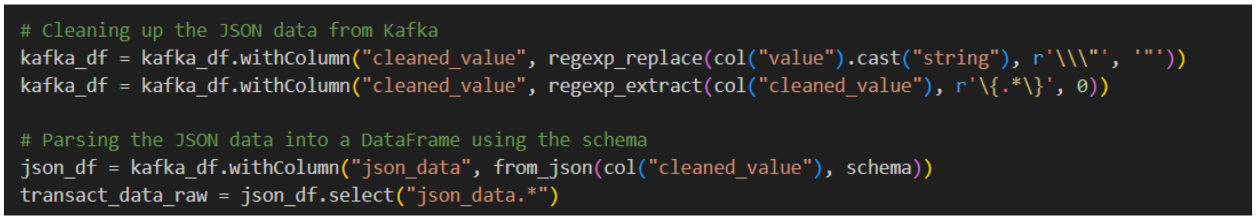
```
kafka_df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("subscribe", "transactions-topic-verified") \
    .load()
```

- **Schema Definition:** The expected schema of the JSON data is defined using `StructType`.



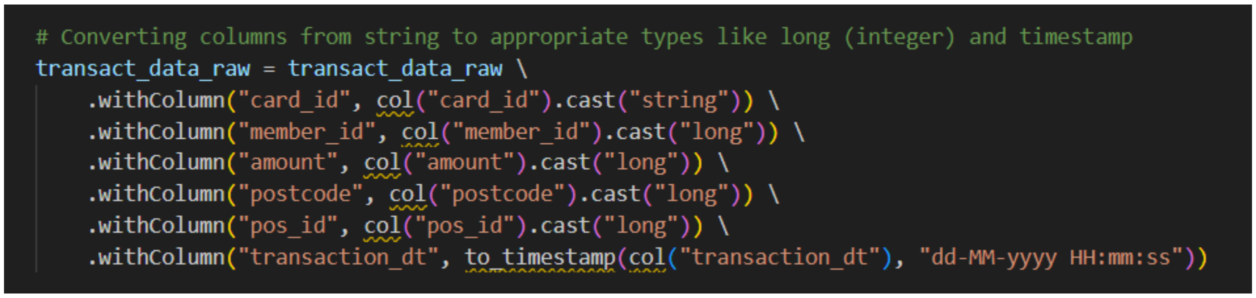
```
schema = StructType([
    StructField("card_id", StringType(), True),
    StructField("member_id", StringType(), True),
    StructField("amount", StringType(), True),
    StructField("postcode", StringType(), True),
    StructField("pos_id", StringType(), True),
    StructField("transaction_dt", StringType(), True)
])
```

- **Data Cleaning and Parsing:** The JSON data is cleaned to remove unwanted characters and then parsed using the defined schema.



```
# Cleaning up the JSON data from Kafka
kafka_df = kafka_df.withColumn("cleaned_value", regexp_replace(col("value").cast("string"), r'\\\"', '"'))
kafka_df = kafka_df.withColumn("cleaned_value", regexp_extract(col("cleaned_value"), r'\{.*\}', 0))

# Parsing the JSON data into a DataFrame using the schema
json_df = kafka_df.withColumn("json_data", from_json(col("cleaned_value"), schema))
transact_data_raw = json_df.select("json_data.*")
```

- **Data Type Conversion:** Columns are converted from string types to appropriate types like `long` and `timestamp`.

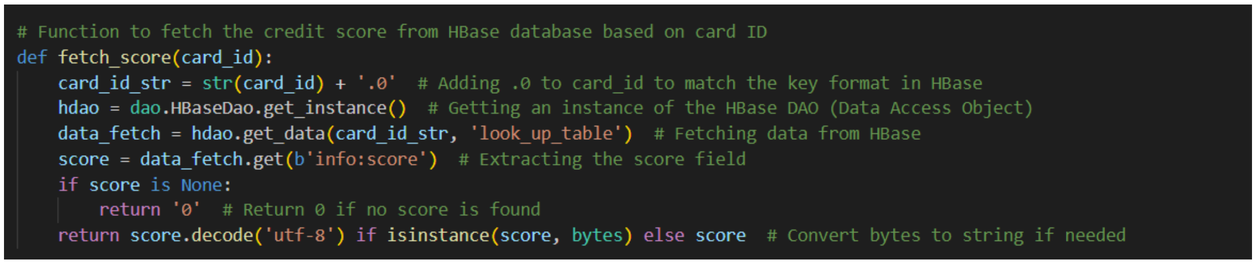


```
# Converting columns from string to appropriate types like long (integer) and timestamp
transact_data_raw = transact_data_raw \
    .withColumn("card_id", col("card_id").cast("string")) \
    .withColumn("member_id", col("member_id").cast("long")) \
    .withColumn("amount", col("amount").cast("long")) \
    .withColumn("postcode", col("postcode").cast("long")) \
    .withColumn("pos_id", col("pos_id").cast("long")) \
    .withColumn("transaction_dt", to_timestamp(col("transaction_dt"), "dd-MM-yyyy HH:mm:ss"))
```

3. User-Defined Functions (UDFs) and Data Enrichment

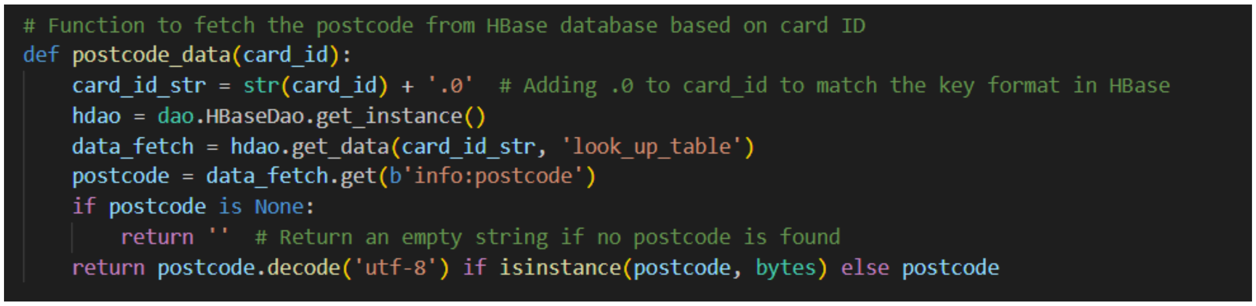
- **HBase Data Retrieval Functions:** Functions are defined to retrieve data from the HBase database, including credit scores, postcodes, upper control limits (UCL), and last transaction dates.

# HBase data fetching logic for credit score



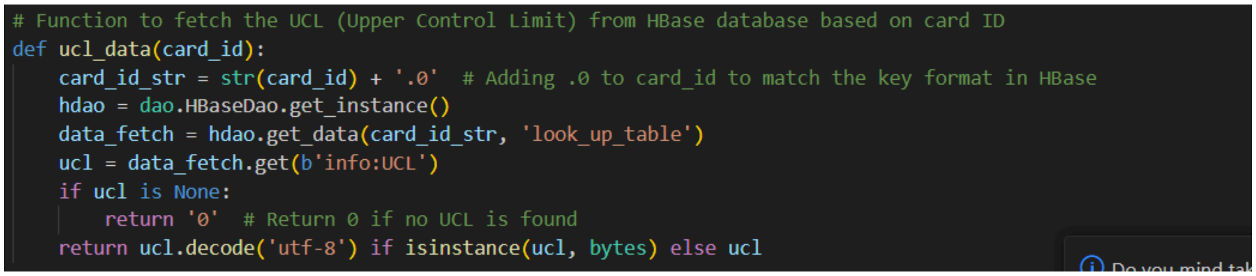
```python
# Function to fetch the credit score from HBase database based on card ID
def fetch_score(card_id):
    card_id_str = str(card_id) + '.0'  # Adding .0 to card_id to match the key format in HBase
    hdao = dao.HBaseDao.get_instance()  # Getting an instance of the HBase DAO (Data Access Object)
    data_fetch = hdao.get_data(card_id_str, 'look_up_table')  # Fetching data from HBase
    score = data_fetch.get(b'info:score')  # Extracting the score field
    if score is None:
        return '0'  # Return 0 if no score is found
    return score.decode('utf-8') if isinstance(score, bytes) else score  # Convert bytes to string if needed
```

# HBase data fetching logic for postcode



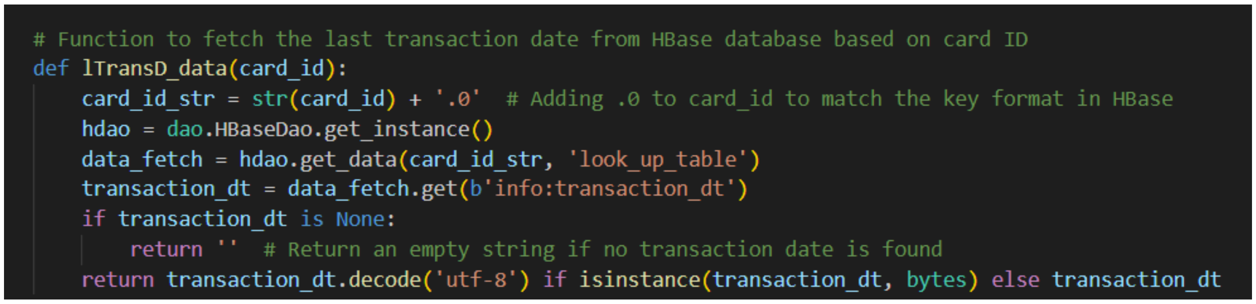
```python
# Function to fetch the postcode from HBase database based on card ID
def postcode_data(card_id):
    card_id_str = str(card_id) + '.0'  # Adding .0 to card_id to match the key format in HBase
    hdao = dao.HBaseDao.get_instance()
    data_fetch = hdao.get_data(card_id_str, 'look_up_table')
    postcode = data_fetch.get(b'info:postcode')
    if postcode is None:
        return ''  # Return an empty string if no postcode is found
    return postcode.decode('utf-8') if isinstance(postcode, bytes) else postcode
```

# HBase data fetching logic for UCL



```python
# Function to fetch the UCL (Upper Control Limit) from HBase database based on card ID
def ucl_data(card_id):
    card_id_str = str(card_id) + '.0'  # Adding .0 to card_id to match the key format in HBase
    hdao = dao.HBaseDao.get_instance()
    data_fetch = hdao.get_data(card_id_str, 'look_up_table')
    ucl = data_fetch.get(b'info:UCL')
    if ucl is None:
        return '0'  # Return 0 if no UCL is found
    return ucl.decode('utf-8') if isinstance(ucl, bytes) else ucl
```

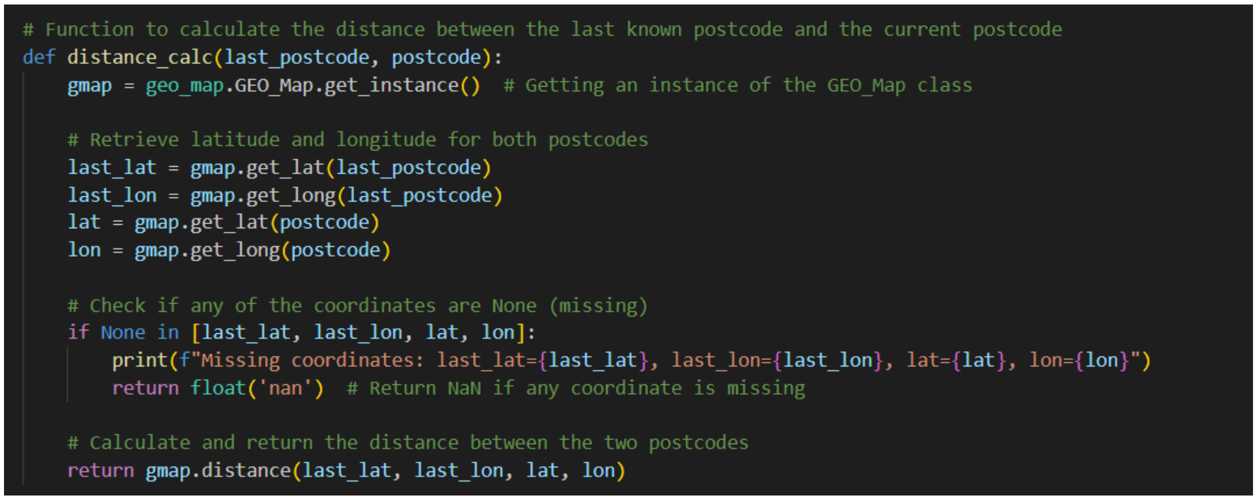
# HBase data fetching logic for last transaction date



```python
# Function to fetch the last transaction date from HBase database based on card ID
def lTransD_data(card_id):
    card_id_str = str(card_id) + '.0'  # Adding .0 to card_id to match the key format in HBase
    hdao = dao.HBaseDao.get_instance()
    data_fetch = hdao.get_data(card_id_str, 'look_up_table')
    transaction_dt = data_fetch.get(b'info:transaction_dt')
    if transaction_dt is None:
        return ''  # Return an empty string if no transaction date is found
    return transaction_dt.decode('utf-8') if isinstance(transaction_dt, bytes) else transaction_dt
```

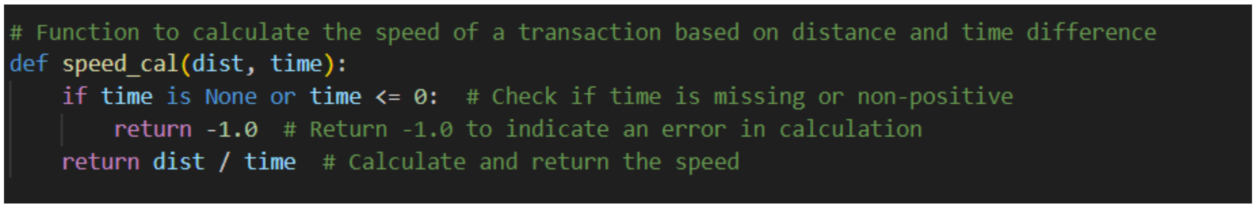
- **Distance and Speed Calculation:** Functions are defined to calculate the distance between postcodes and the speed of transactions based on the distance and time difference.

```
# Distance calculation logic
```



```python
# Function to calculate the distance between the last known postcode and the current postcode
def distance_calc(last_postcode, postcode):
    gmap = geo_map.GEO_Map.get_instance()  # Getting an instance of the GEO_Map class

    # Retrieve latitude and longitude for both postcodes
    last_lat = gmap.get_lat(last_postcode)
    last_lon = gmap.get_long(last_postcode)
    lat = gmap.get_lat(postcode)
    lon = gmap.get_long(postcode)

    # Check if any of the coordinates are None (missing)
    if None in [last_lat, last_lon, lat, lon]:
        print(f"Missing coordinates: last_lat={last_lat}, last_lon={last_lon}, lat={lat}, lon={lon}")
        return float('nan')  # Return NaN if any coordinate is missing

    # Calculate and return the distance between the two postcodes
    return gmap.distance(last_lat, last_lon, lat, lon)
```

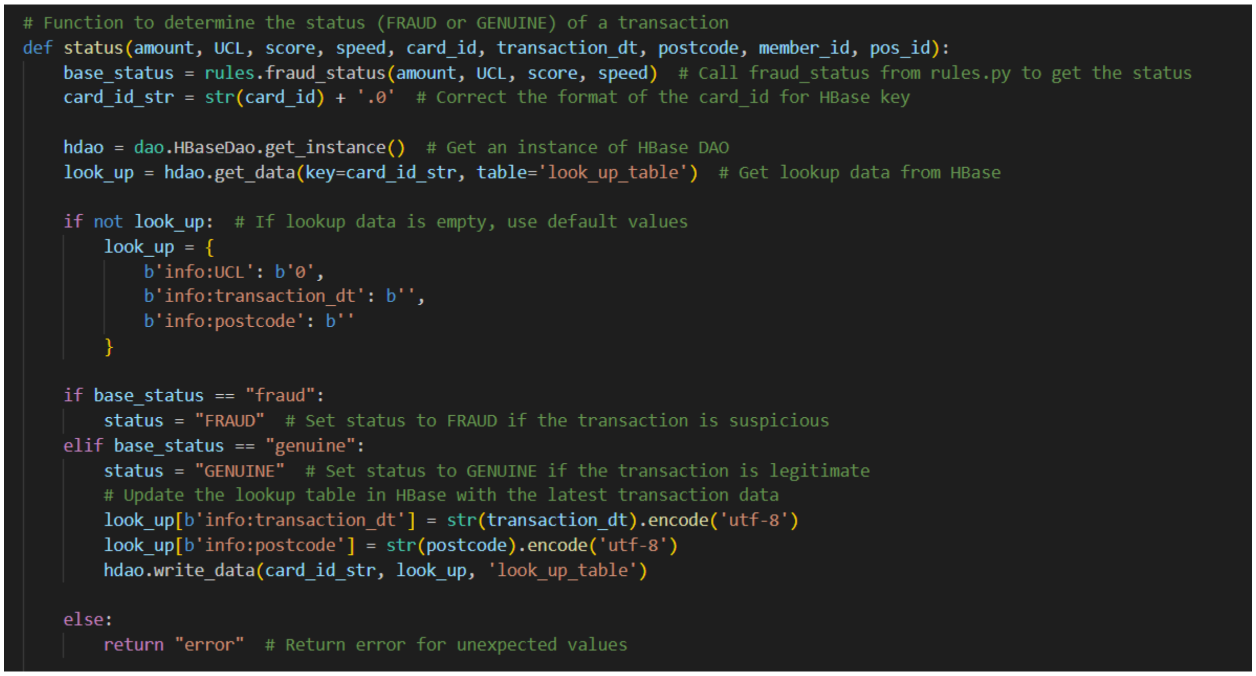
```
# Speed calculation logic
```



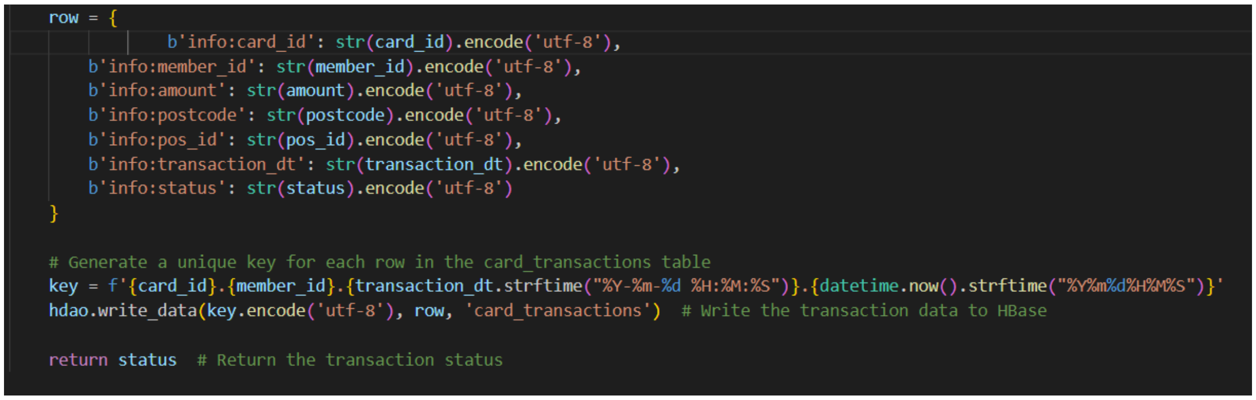
```python
# Function to calculate the speed of a transaction based on distance and time difference
def speed_cal(dist, time):
    if time is None or time <= 0:  # Check if time is missing or non-positive
        return -1.0  # Return -1.0 to indicate an error in calculation
    return dist / time  # Calculate and return the speed
```

- **Fraud Detection Status:** The `status` function determines whether a transaction is "FRAUD" or "GENUINE" based on various factors such as amount, UCL, credit score, and speed.

```
# Fraud detection and HBase update logic
```



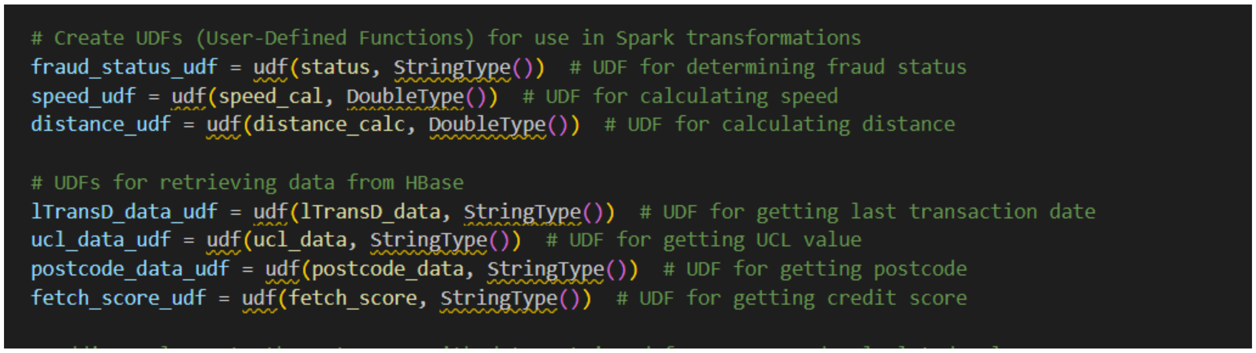
```python
# Function to determine the status (FRAUD or GENUINE) of a transaction
def status(amount, UCL, score, speed, card_id, transaction_dt, postcode, member_id, pos_id):
    base_status = rules.fraud_status(amount, UCL, score, speed)  # Call fraud_status from rules.py to get the status
    card_id_str = str(card_id) + '.0'  # Correct the format of the card_id for HBase key

    hdao = dao.HBaseDao.get_instance()  # Get an instance of HBase DAO
    look_up = hdao.get_data(key=card_id_str, table='look_up_table')  # Get lookup data from HBase

    if not look_up:  # If lookup data is empty, use default values
        look_up = {
            b'info:UCL': b'0',
            b'info:transaction_dt': b'',
            b'info:postcode': b''
        }

    if base_status == "fraud":
        status = "FRAUD"  # Set status to FRAUD if the transaction is suspicious
    elif base_status == "genuine":
        status = "GENUINE"  # Set status to GENUINE if the transaction is legitimate
        # Update the lookup table in HBase with the latest transaction data
        look_up[b'info:transaction_dt'] = str(transaction_dt).encode('utf-8')
        look_up[b'info:postcode'] = str(postcode).encode('utf-8')
        hdao.write_data(card_id_str, look_up, 'look_up_table')

    else:
        return "error"  # Return error for unexpected values
```



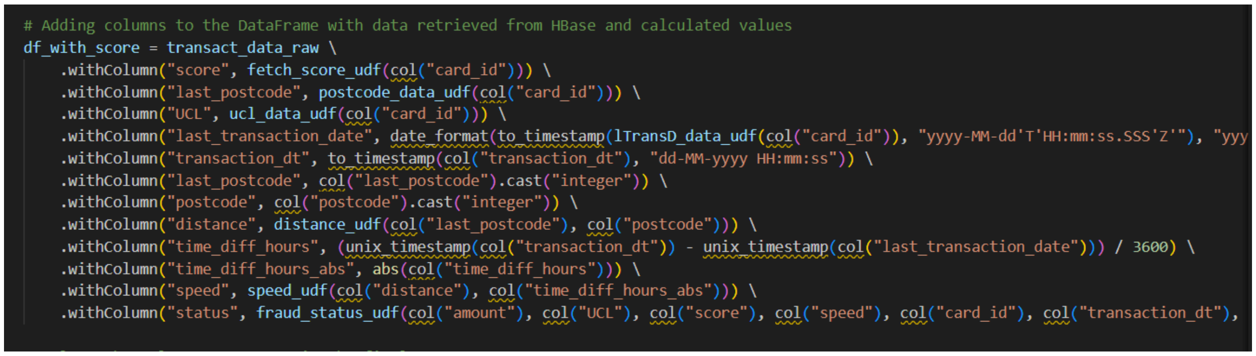
```python
    row = {
                b'info:card_id': str(card_id).encode('utf-8'),
        b'info:member_id': str(member_id).encode('utf-8'),
        b'info:amount': str(amount).encode('utf-8'),
        b'info:postcode': str(postcode).encode('utf-8'),
        b'info:pos_id': str(pos_id).encode('utf-8'),
        b'info:transaction_dt': str(transaction_dt).encode('utf-8'),
        b'info:status': str(status).encode('utf-8')
    }

    # Generate a unique key for each row in the card_transactions table
    key = f'{card_id}.{member_id}.{transaction_dt.strftime("%Y-%m-%d %H:%M:%S")}.{datetime.now().strftime("%Y%m%d%H%M%S")}'
    hdao.write_data(key.encode('utf-8'), row, 'card_transactions')  # Write the transaction data to HBase

    return status  # Return the transaction status
```

4. Data Transformation and Streaming Output

- **Applying UDFs:** The raw transaction data is enriched with additional columns by applying the UDFs.

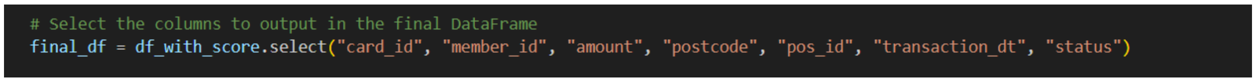


```
# Create UDFs (User-Defined Functions) for use in Spark transformations
fraud_status_udf = udf(status, StringType())  # UDF for determining fraud status
speed_udf = udf(speed_cal, DoubleType())  # UDF for calculating speed
distance_udf = udf(distance_calc, DoubleType())  # UDF for calculating distance

# UDFs for retrieving data from HBase
lTransD_data_udf = udf(lTransD_data, StringType())  # UDF for getting last transaction date
ucl_data_udf = udf(ucl_data, StringType())  # UDF for getting UCL value
postcode_data_udf = udf(postcode_data, StringType())  # UDF for getting postcode
fetch_score_udf = udf(fetch_score, StringType())  # UDF for getting credit score
```



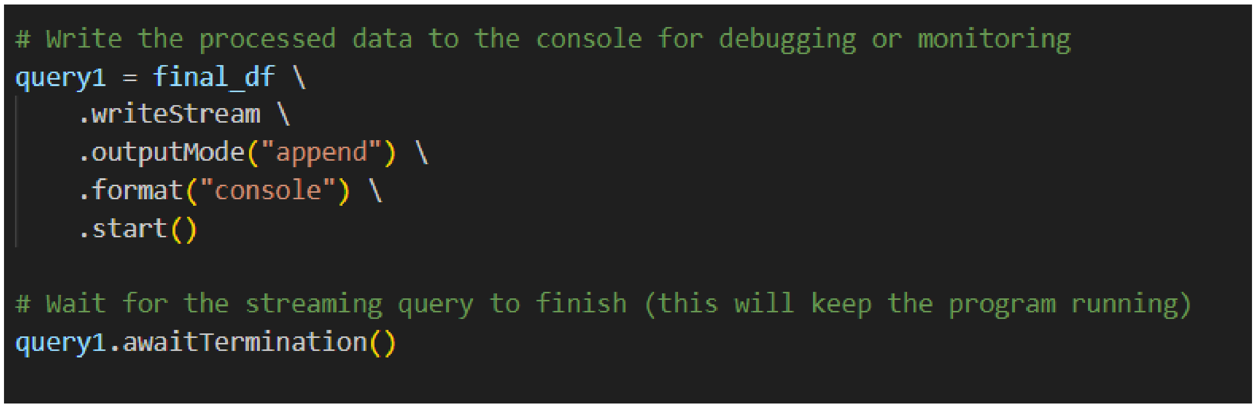
```
# Adding columns to the DataFrame with data retrieved from HBase and calculated values
df_with_score = transact_data_raw \
    .withColumn("score", fetch_score_udf(col("card_id"))) \
    .withColumn("last_postcode", postcode_data_udf(col("card_id"))) \
    .withColumn("UCL", ucl_data_udf(col("card_id"))) \
    .withColumn("last_transaction_date", date_format(to_timestamp(lTransD_data_udf(col("card_id")), "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"), "yyy
    .withColumn("transaction_dt", to_timestamp(col("transaction_dt"), "dd-MM-yyyy HH:mm:ss")) \
    .withColumn("last_postcode", col("last_postcode").cast("integer")) \
    .withColumn("postcode", col("postcode").cast("integer")) \
    .withColumn("distance", distance_udf(col("last_postcode"), col("postcode"))) \
    .withColumn("time_diff_hours", (unix_timestamp(col("transaction_dt")) - unix_timestamp(col("last_transaction_date"))) / 3600) \
    .withColumn("time_diff_hours_abs", abs(col("time_diff_hours"))) \
    .withColumn("speed", speed_udf(col("distance"), col("time_diff_hours_abs"))) \
    .withColumn("status", fraud_status_udf(col("amount"), col("UCL"), col("score"), col("speed"), col("card_id"), col("transaction_dt"),
```

- **Final Data Selection:** Only the relevant columns are selected for output.



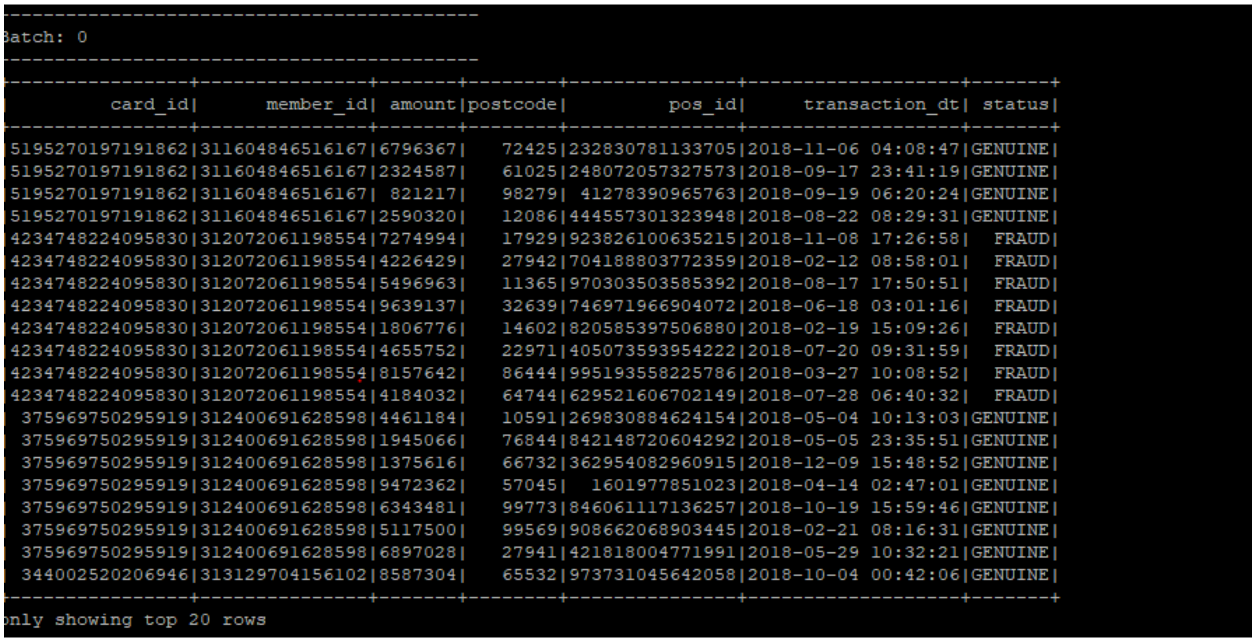
```
# Select the columns to output in the final DataFrame
final_df = df_with_score.select("card_id", "member_id", "amount", "postcode", "pos_id", "transaction_dt", "status")
```

- **Streaming Output:** The processed data is written to the console for monitoring or debugging.

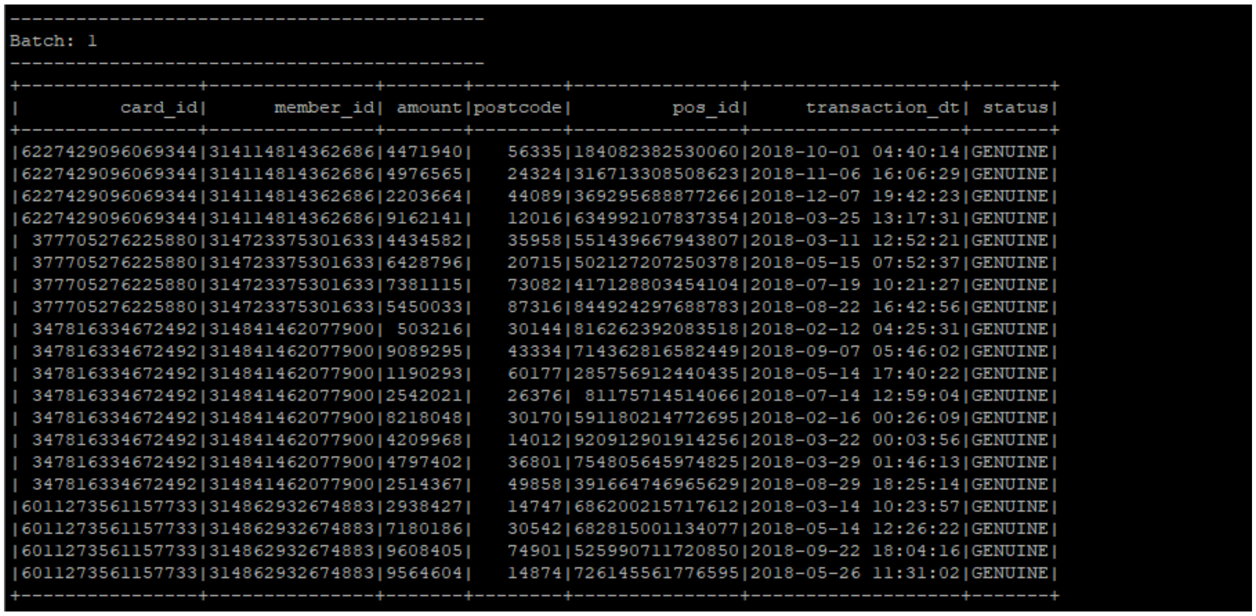


```
# Write the processed data to the console for debugging or monitoring
query1 = final_df \
    .writeStream \
    .outputMode("append") \
    .format("console") \
    .start()

# Wait for the streaming query to finish (this will keep the program running)
query1.awaitTermination()
```

Output of driver.py

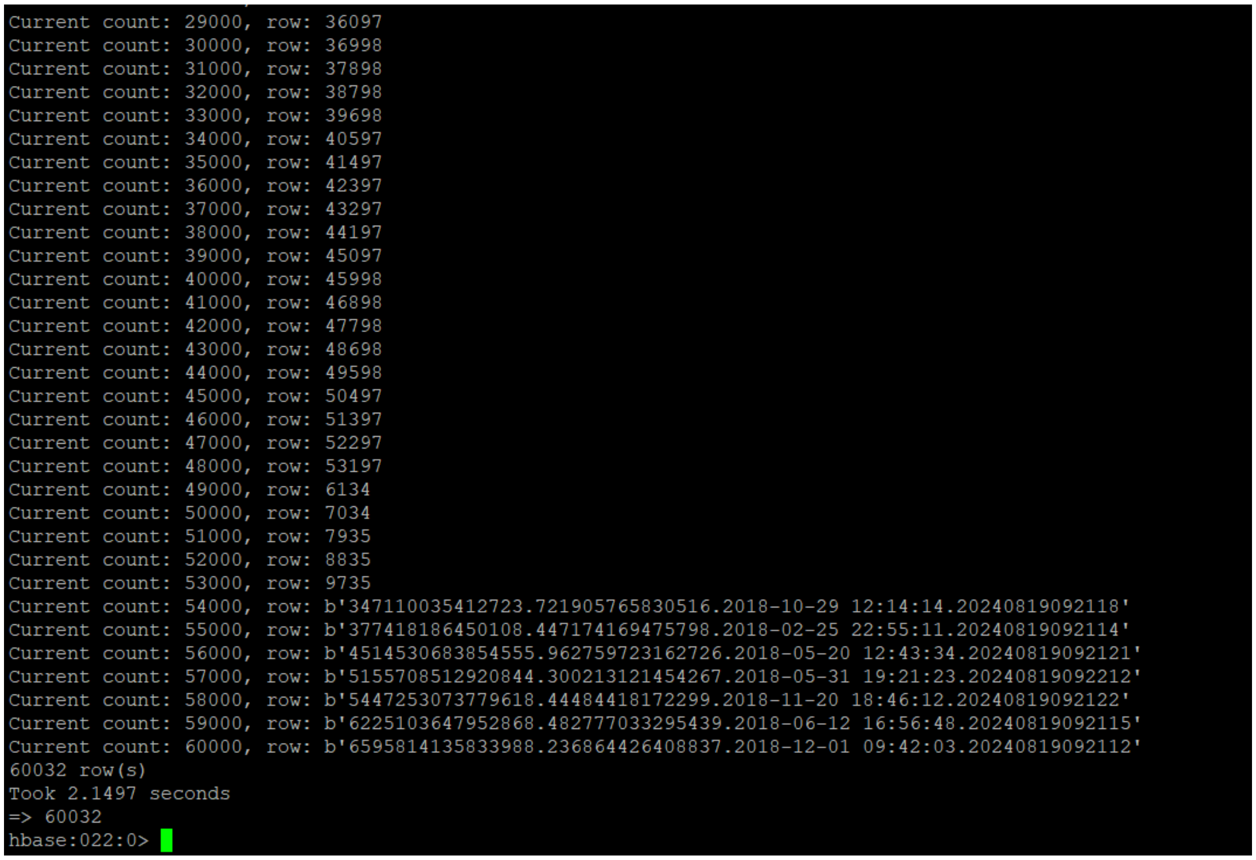


```
-------------------------------------------
atch: 0
-------------------------------------------
+----------------+---------------+-------+--------+---------------+-------------------+-------+
|         card_id|      member_id| amount|postcode|         pos_id|     transaction_dt| status|
+----------------+---------------+-------+--------+---------------+-------------------+-------+
|5195270197191862|311604846516167|6796367|   72425|232830781133705|2018-11-06 04:08:47|GENUINE|
|5195270197191862|311604846516167|2324587|   61025|248072057327573|2018-09-17 23:41:19|GENUINE|
|5195270197191862|311604846516167| 821217|   98279| 41278390965763|2018-09-19 06:20:24|GENUINE|
|5195270197191862|311604846516167|2590320|   12086|444557301323948|2018-08-22 08:29:31|GENUINE|
|4234748224095830|312072061198554|7274994|   17929|923826100635215|2018-11-08 17:26:58|  FRAUD|
|4234748224095830|312072061198554|4226429|   27942|704188803772359|2018-02-12 08:58:01|  FRAUD|
|4234748224095830|312072061198554|5496963|   11365|970303503585392|2018-08-17 17:50:51|  FRAUD|
|4234748224095830|312072061198554|9639137|   32639|746971966904072|2018-06-18 03:01:16|  FRAUD|
|4234748224095830|312072061198554|1806776|   14602|820585397506880|2018-02-19 15:09:26|  FRAUD|
|4234748224095830|312072061198554|4655752|   22971|405073593954222|2018-07-20 09:31:59|  FRAUD|
|4234748224095830|312072061198554|8157642|   86444|995193558225786|2018-03-27 10:08:52|  FRAUD|
|4234748224095830|312072061198554|4184032|   64744|629521606702149|2018-07-28 06:40:32|  FRAUD|
| 375969750295919|312400691628598|4461184|   10591|269830884624154|2018-05-04 10:13:03|GENUINE|
| 375969750295919|312400691628598|1945066|   76844|842148720604292|2018-05-05 23:35:51|GENUINE|
| 375969750295919|312400691628598|1375616|   66732|362954082960915|2018-12-09 15:48:52|GENUINE|
| 375969750295919|312400691628598|9472362|   57045|  1601977851023|2018-04-14 02:47:01|GENUINE|
| 375969750295919|312400691628598|6343481|   99773|846061117136257|2018-10-19 15:59:46|GENUINE|
| 375969750295919|312400691628598|5117500|   99569|908662068903445|2018-02-21 08:16:31|GENUINE|
| 375969750295919|312400691628598|6897028|   27941|421818004771991|2018-05-29 10:32:21|GENUINE|
| 344002520206946|313129704156102|8587304|   65532|97373104564205 8|2018-10-04 00:42:06|GENUINE|
+----------------+---------------+-------+--------+---------------+-------------------+-------+
nly showing top 20 rows
```

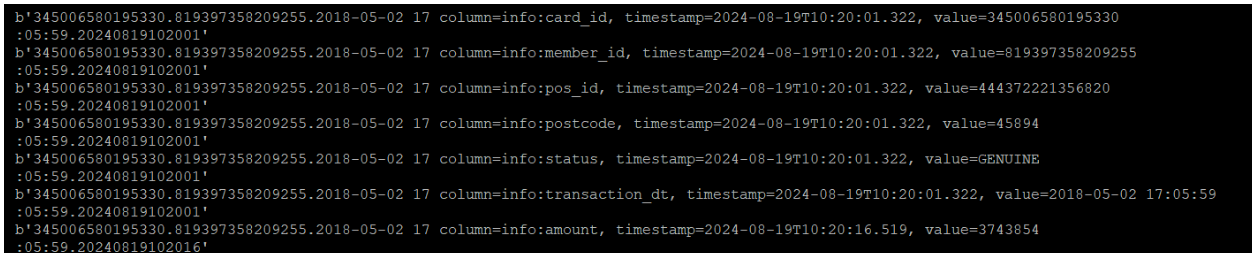


```
-------------------------------------------
Batch: 1
-------------------------------------------
+----------------+---------------+-------+--------+---------------+-------------------+-------+
|         card_id|      member_id| amount|postcode|         pos_id|     transaction_dt| status|
+----------------+---------------+-------+--------+---------------+-------------------+-------+
|6227429096069344|314114814362686|4471940|   56335|184082382530060|2018-10-01 04:40:14|GENUINE|
|6227429096069344|314114814362686|4976565|   24324|316713308508623|2018-11-06 16:06:29|GENUINE|
|6227429096069344|314114814362686|2203664|   44089|369295688877266|2018-12-07 19:42:23|GENUINE|
|6227429096069344|314114814362686|9162141|   12016|634992107837354|2018-03-25 13:17:31|GENUINE|
| 377705276225880|314723375301633|4434582|   35958|551439667943807|2018-03-11 12:52:21|GENUINE|
| 377705276225880|314723375301633|6428796|   20715|502127207250378|2018-05-15 07:52:37|GENUINE|
| 377705276225880|314723375301633|7381115|   73082|417128803454104|2018-07-19 10:21:27|GENUINE|
| 377705276225880|314723375301633|5450033|   87316|844924297688783|2018-08-22 16:42:56|GENUINE|
| 347816334672492|314841462077900| 503216|   30144|816262392083518|2018-02-12 04:25:31|GENUINE|
| 347816334672492|314841462077900|9089295|   43334|714362816582449|2018-09-07 05:46:02|GENUINE|
| 347816334672492|314841462077900|1190293|   60177|285756912440435|2018-05-14 17:40:22|GENUINE|
| 347816334672492|314841462077900|2542021|   26376| 81175714514066|2018-07-14 12:59:04|GENUINE|
| 347816334672492|314841462077900|8218048|   30170|591180214772695|2018-02-16 00:26:09|GENUINE|
| 347816334672492|314841462077900|4209968|   14012|920912901914256|2018-03-22 00:03:56|GENUINE|
| 347816334672492|314841462077900|4797402|   36801|754805645974825|2018-03-29 01:46:13|GENUINE|
| 347816334672492|314841462077900|2514367|   49858|391664746965629|2018-08-29 18:25:14|GENUINE|
|6011273561157733|314862932674883|2938427|   14747|686200215717612|2018-03-14 10:23:57|GENUINE|
|6011273561157733|314862932674883|7180186|   30542|682815001134077|2018-05-14 12:26:22|GENUINE|
|6011273561157733|314862932674883|9608405|   74901|525990711720850|2018-09-22 18:04:16|GENUINE|
|6011273561157733|314862932674883|9564604|   14874|726145561776595|2018-05-26 11:31:02|GENUINE|
+----------------+---------------+-------+--------+---------------+-------------------+-------+
```

The count of card transactions in the card_transactions table after executing the driver program.



```
Current count: 29000, row: 36097
Current count: 30000, row: 36998
Current count: 31000, row: 37898
Current count: 32000, row: 38798
Current count: 33000, row: 39698
Current count: 34000, row: 40597
Current count: 35000, row: 41497
Current count: 36000, row: 42397
Current count: 37000, row: 43297
Current count: 38000, row: 44197
Current count: 39000, row: 45097
Current count: 40000, row: 45998
Current count: 41000, row: 46898
Current count: 42000, row: 47798
Current count: 43000, row: 48698
Current count: 44000, row: 49598
Current count: 45000, row: 50497
Current count: 46000, row: 51397
Current count: 47000, row: 52297
Current count: 48000, row: 53197
Current count: 49000, row: 6134
Current count: 50000, row: 7034
Current count: 51000, row: 7935
Current count: 52000, row: 8835
Current count: 53000, row: 9735
Current count: 54000, row: b'347110035412723.721905765830516.2018-10-29 12:14:14.20240819092118'
Current count: 55000, row: b'377418186450108.447174169475798.2018-02-25 22:55:11.20240819092114'
Current count: 56000, row: b'4514530683854555.962759723162726.2018-05-20 12:43:34.20240819092121'
Current count: 57000, row: b'5155708512920844.300213121454267.2018-05-31 19:21:23.20240819092212'
Current count: 58000, row: b'5447253073779618.44484418172299.2018-11-20 18:46:12.20240819092122'
Current count: 59000, row: b'6225103647952868.482777033295439.2018-06-12 16:56:48.20240819092115'
Current count: 60000, row: b'6595814135833988.236864426408837.2018-12-01 09:42:03.20240819092112'
60032 row(s)
Took 2.1497 seconds
=> 60032
hbase:022:0>
```



```
 b'345006580195330.819397358209255.2018-05-02 17 column=info:card_id, timestamp=2024-08-19T10:20:01.322, value=345006580195330
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:member_id, timestamp=2024-08-19T10:20:01.322, value=819397358209255
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:pos_id, timestamp=2024-08-19T10:20:01.322, value=444372221356820
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:postcode, timestamp=2024-08-19T10:20:01.322, value=45894
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:status, timestamp=2024-08-19T10:20:01.322, value=GENUINE
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:transaction_dt, timestamp=2024-08-19T10:20:01.322, value=2018-05-02 17:05:59
 :05:59.20240819102001'
 b'345006580195330.819397358209255.2018-05-02 17 column=info:amount, timestamp=2024-08-19T10:20:16.519, value=3743854
 :05:59.20240819102016'
```

Conclusion

This script processes real-time transaction data to detect potential fraud by comparing each transaction against historical data stored in Look_up_table in hbase. The results are then streamed to the console for monitoring. The use of UDFs enables custom logic for fraud detection, making the system highly adaptable to various business rules.