Problem Statement

Imagine you are working for a leading credit card company called **'Cred Financials'**. The company continuously monitors its customers' credit card transactions, be it in any part of the world, to discover and dismiss fraudulent ones. The company also has a strong support team to address customer issues and queries.

You, as a big data engineer, must architect and build a solution to cater to the following requirements:

1. Fraud detection solution: This is a feature to detect fraudulent transactions, wherein once a cardmember swipes his/her card for payment, the transaction should be classified as fraudulent or authentic based on a set of predefined rules. If fraud is detected, then the transaction must be declined. Please note that incorrectly classifying a transaction as fraudulent will incur huge losses to the company and also provoke negative consumer sentiment.

2. Customers' information: The relevant information about the customers' needs to be continuously updated on a platform from where the customer support team can retrieve relevant information in real time to resolve customer complaints and queries.

Artifacts to be considered:

The following tables containing data come into consideration for this problem:

card_member (The cardholder data is added to/updated in this table by a third-party service)

Column Description:

card_id – Card number,

member_id – 15-digit member ID of the cardholder,

member_joining_dt – Date and time of joining of a new member,

card_purchase_dt – Date when the card was purchased,

country – Country in which the card was purchased,

city – City in which the card was purchased

card_transactions (All incoming transactions(fraud/genuine) swiped at POS terminals are stored in this table. Earlier the transactions were classified as fraud or genuine in a traditional way. However, with an explosive surge in the number of transactions, a Big Data solution is needed to authenticate the incoming transactions and enter the transaction data accordingly):

Column Description:

card_id – Card number,

member_id – 15-digit member ID of the cardholder,

amount – Amount swiped with respect to the card_id,

postcode – Zip code at which this card was swiped (marking the location of an event),

pos_id – Merchant's POS terminal ID, using which the card has been swiped,

transaction_dt – date and time of the transaction event,

status – Whether transaction was approved or not, with Genuine/Fraud value

member_score (The member credit score data is added to / updated in this table by a third-party service):

member_id – 15-digit member ID who has this card,

score – The score assigned to a member defining his/her credit history, generated by upstream systems.

SOLUTION STEPS:

1. **Setup directory in HDFS for the project. After connecting to ec2 instance via ec2-user, switch to root user and then to hdfs user. Create directory and change its ownership and then exit from hdfs user and then exit from root user and this will bring back to ec2-user.**

2. **Download card_transactions and zipCodePosId csv's from resources section in the capstone project and transfer it to ec2 instance via WinSCP. Copy both the files to HDFS on the location created above.**

3. **load the data and create table/s in the NoSQL database**
    a. Create new database by the name credit_card_fraud_detection
    b. Set some parameters for hive session
    c. Create external table card_transactions_ext table which will point to HDFS location
    d. Create table card_transactions_orc
    e. Load data in card_transaction_orc while casting timestamp format for transaction_dt column
    f. Verify transaction_dt and year in card_transactions_orc
    g. Create card_transactions_hbase as hive-hbase integrated table which will be visible from Hbase as well.
    h. Load data in card_transactions_hbase.
    i. Check for some data in card_transactions_hbase
    j. Create lookup_data_hbase as hive-hbase integrated table.
    k. In Hbase check details of card_transaction_hive
    l. In Hbase check details of lookup_data_hive integrated tables
    m. In Hbase, alter lookup_data_hive table and set VERSIONS to 10 for lookup_transaction_family
    n. Check details of lookup_data_hive and confirm that VERSIONS is set to 10 for lookup_transaction_family.

4. **Ingest relevant data from AWS RDS to Sqoop.**
    a. Create external table card_member_ext which will point to HDFS location.
    b. Create external table member_score_ext which will point to HDFS location
    c. Create card_member_orc table
    d. Create member_score_orc table
    e. Load data into card_member_orc
    f. Load data into member_score_orc
    g. Verify some data in card_member_orc table
    h. Verify some data in member_score_orc table

5. **Calculate the moving average and standard deviation of the last 10 transactions for each card_id for the data present in Hadoop and NoSQL database. If the total number of transactions for a particular card_id is less than 10, then calculate the parameters based on the total number of records available for that card_id.**
    a. Create table ranked_card_transactions_orc to store 10 transaction for each card_id.
    b. Create table card_ucl_orc to store UCL values for each card_id
    c. Load data in ranked_card_transactions_orc table.
    d. Load data in card_ucl_orc table
    e. Load data in lookup_data_hbase table.
    f. Verify count in lookup_data_hbase table
    g. Verify some data in lookup_data_hbase table
    h. Check count in lookup_data_hive table
    i. Check data in lookup_data_hive table

6. **Verify data:**
    a. Check data in Hbase lookup_data_hive_table
    b. Check data for a particular card_id, see multiple versions for postcode and transaction_dt
    c. Check data for a particular card_id, verify that there should not be any multiple versions for ucl and score
    d. Check data for a particular card_id