Credit Card Fraud

SPARK STREAMING & SPARK SQL

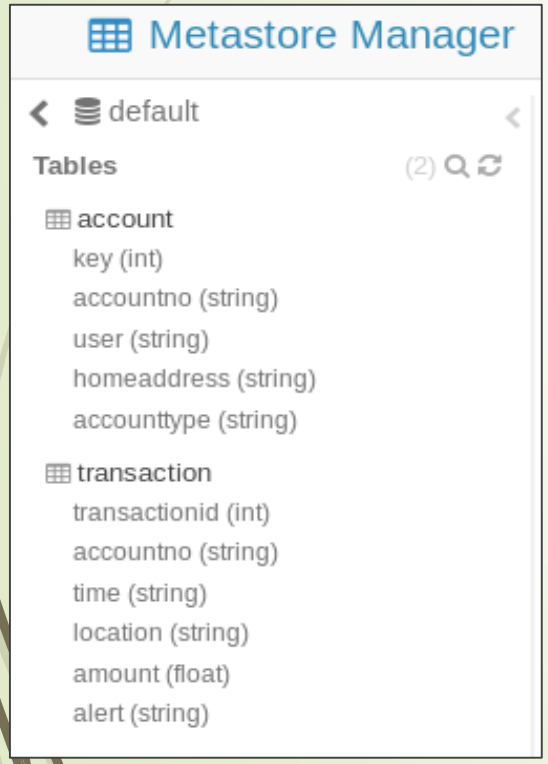
Brief Project Overview

- This project analysis & examine whether the new transaction is “enriched” or “fraud”

- This application is developed by JAVA 8 and SPARK API
  - SPARK STREAMING
  - SPARK SQL Integrate HIVE
  - Data Streaming with NetCat
  - Cloudera & Hadoop

Database Structure

- **Account**: Contains Bank Account information as AccountNo, HomeAddress, UserName, AccountType
- **Transaction**: For keeping track of Bank Transactions as TransactionId, AccountNo, Time, Location, Amount, Alert



- Account

SAMPLE

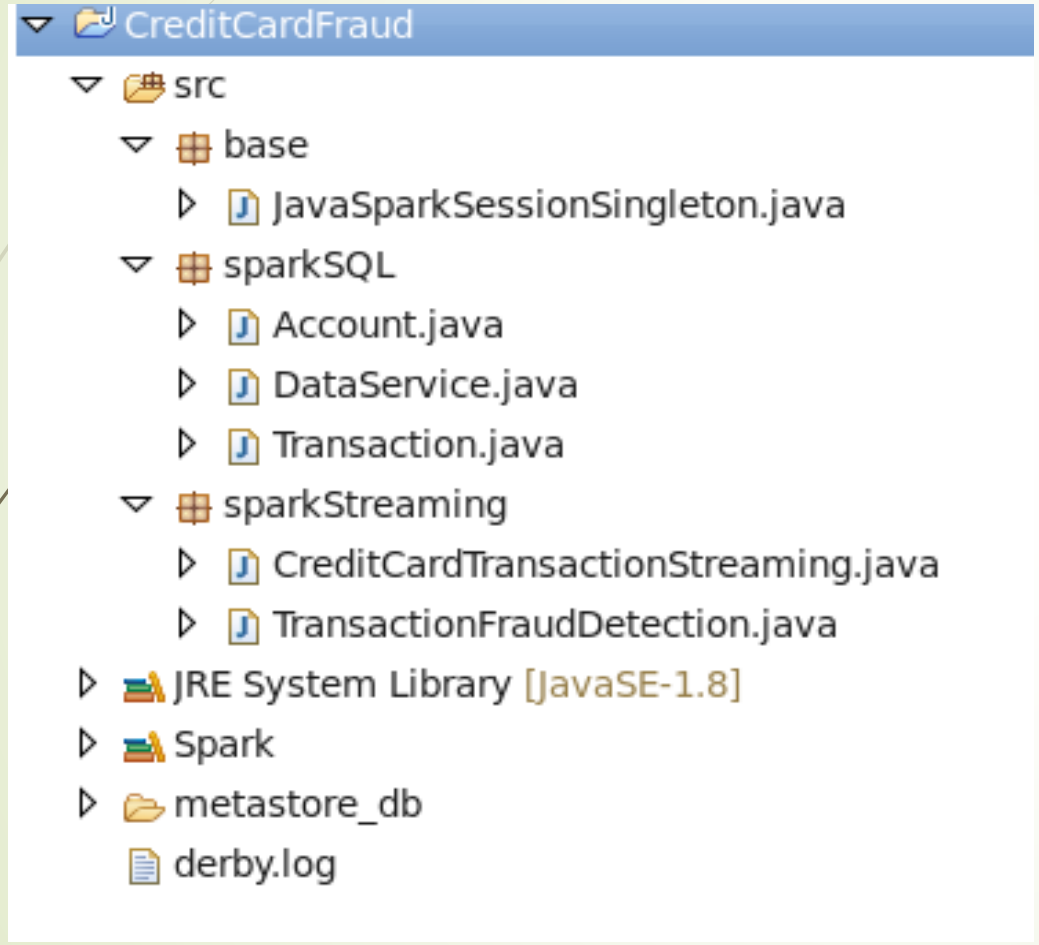
	account.key	account.accountno	account.user	account.homeaddress	account.accounttype
1	1	6771926475977940	Smith	1307 Amber Butterfly Freeway|Woods And Irons|New Jersey|US	CreditCard
2	2	3567052837447110	John	4298 Broad Wood|Haskingsville|Rhode Island|US	MasterCard
3	3	30128306034706	Bob	1952 Lazy Lookout|Golden Hill|California|US	DebitCard

- Transaction

SAMPLE

	transaction.transactionid	transaction.accountno	transaction.time	transaction.location	transaction.amount	transaction.alert
1	1	6771926475977940	5/7/2016	5391 Cozy Knoll|Wolf Creek|Missouri|US	73.80000305175781	""
2	2	3567052837447110	6/22/2016	6850 Jagged Campus|Adams Morgan|Nevada|US	496.1000061035156	""
3	3	30128306034706	8/18/2016	3828 Red Trail|Squab Hollow|Nunavut|CA	578.8900146484375	""

Solution Structure



- **DataSevice**:

  SparkSQL implementation which handle to load and Insert data to Hive

- **Bean Classes**: **Account** & **Transaction**

- **CreditCardTransactionStream**:

  SparkStream implementation which handle to receive/analysis streaming data.

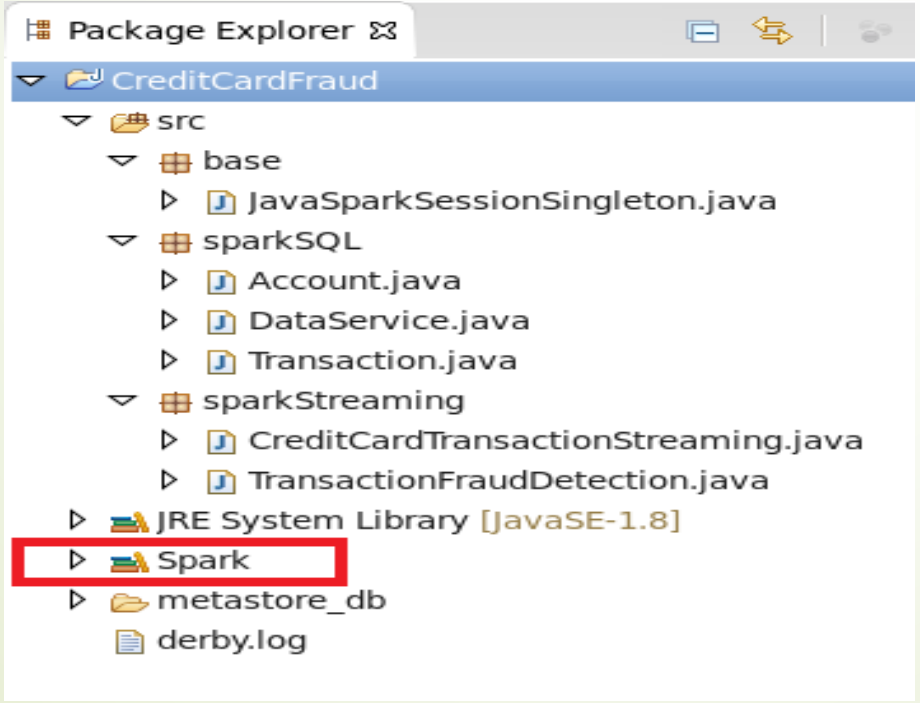
  Integrate with **DataService** and **TransactionFraudDectection** Engine to analysis data

Environment Preparation

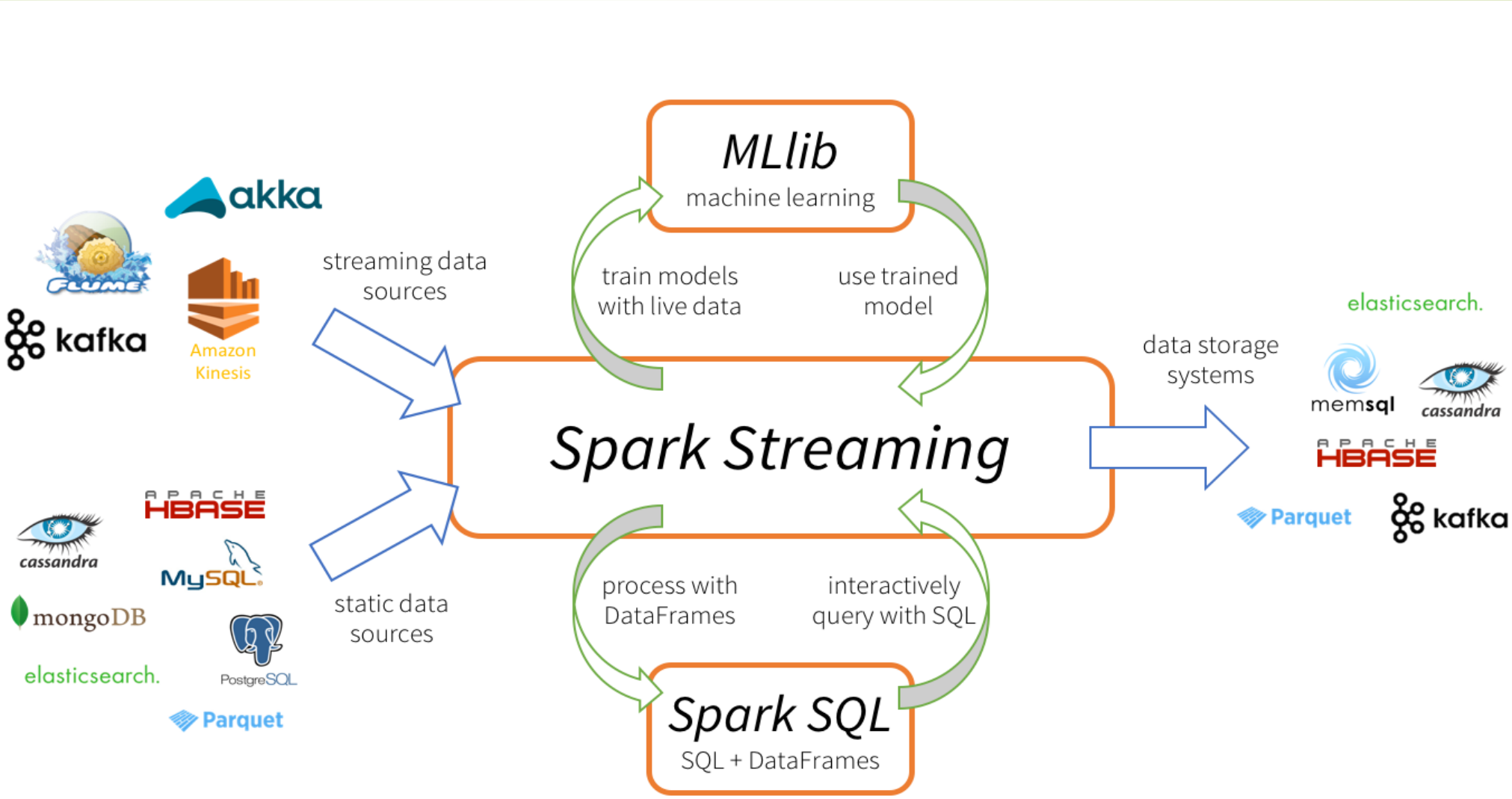
- Upgrade Spark version:

```
[cloudera@quickstart ~]$ su
Password:
[root@quickstart cloudera]# sudo yum install spark-core spark-master spark-worker spark-python
```

- References **ONLY** SPARK library

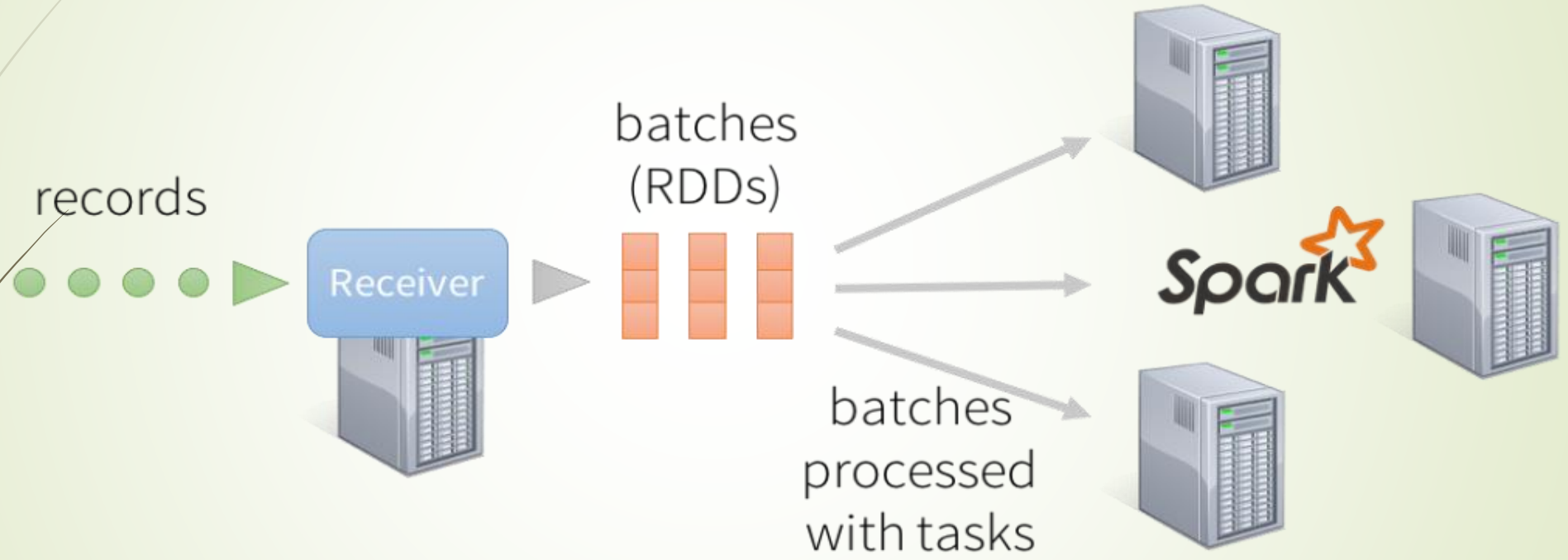


Spark Streaming vs Spark SQL



Spark Streaming

discretized stream processing



records processed in batches with short tasks
each batch is a RDD (partitioned dataset)

Spark Session and Stream Receiver Configuration

```
// Create the context with a 1 second batch size
SparkConf sparkConf = new SparkConf()
        .setAppName("CreditCardTransaction").setMaster("local[2]")
        .set("spark.executor.memory", "1g");

@SuppressWarnings("resource")
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf,
        Durations.seconds(20));

// Create a JavaReceiverInputDStream on target ip:port and count the
// words in input stream of \n delimited text (eg. generated by 'nc')
// Note that no duplication in storage level only for running locally.
// Replication necessary in distributed scenario for fault tolerance.
JavaReceiverInputDStream<String> lines = ssc.socketTextStream(args[0],
        Integer.parseInt(args[1]), StorageLevels.MEMORY_AND_DISK);
```

Convert Single Line of Stream Data to Transaction Bean

```
JavaDStream<Transaction> transactions = lines
        .map(new Function<String, Transaction>() {
            private static final long serialVersionUID = 1L;

            @Override
            public Transaction call(String x) throws Exception {
                String[] splits = SPACE.split(x);
                if (splits.length < 6)
                    return null;

                return new Transaction(Integer.parseInt(splits[0]),
                        splits[1], splits[2], splits[3], Float.parseFloat(splits[4]), splits[5]);
            }
        });
```

Transform and Analysis RDD Data

```
// Convert RDDs of the words DStream to DataFrame and run SQL query
transactions.foreachRDD((rdd, time) -> {
    // Handle partitions empty
        if (rdd.partitions().isEmpty()) {
            return;
        }

        DataService instance = DataService.getInstance(rdd.context());
        TransactionFraudDetection pc = new TransactionFraudDetection();
        HashMap<String, Account> accountHash = new HashMap<>();
        HashMap<String, List<Transaction>> transHash = new HashMap<>();
        HashMap<Integer, Transaction> updateList = new HashMap<>();
        // Loop to each trans
        rdd.collect().forEach( (trans) -> {
            Account acc = null;
            List<Transaction> recentTrans = null;

            if (!accountHash.containsKey(trans.getAccountNo())) {
                // Get Account
                acc = instance.getAccount(trans.getAccountNo());
                accountHash.put(trans.getAccountNo(), acc);

                // Get List of most transaction
                recentTrans = instance.getRecentTransactions(trans.getAccountNo());
                transHash.put(trans.getAccountNo(),recentTrans);
            } else {
                acc = accountHash.get(trans.getAccountNo());
                recentTrans = transHash.get(trans.getAccountNo());
            }
            // Calculate alert
            String alert = pc.calcTotalPossibility(acc,trans, recentTrans);
            trans.setAlert(alert);
            // Add transaction in Hash
            updateList.put(trans.getTransactionId(),trans);
        });
        // Convert back to RDD and ready for saving
        JavaRDD<Transaction> updateAlerTrans = rdd.map(x -> {
            return updateList.get(x.getTransactionId());
        });

        // Insert transaction to Transaction table
        instance.insertNewTransaction(updateAlerTrans);
```


ACADGILD
QUERYING HIVE TABLES
USING SPARK
HIVE
Spark

Spark SQL Configuration

```
SparkConf sparkConf = new SparkConf()
                .setMaster("local[2]")
                .set("hive.metastore.warehouse.dir", "file:/user/hive/warehouse")
                .set("hive.metastore.uris", "thrift://127.0.0.1:9083");

spark = SparkSession
        .builder()
        .config(sparkConf)
        .enableHiveSupport()
        .getOrCreate();

// Create if not exist for Account Table
spark.sql("CREATE TABLE IF NOT EXISTS Account (key INT, AccountNo STRING, User STRING, HomeAddress STRING, AccountType STRING)");
// Create if not exist for Transaction Table
spark.sql("CREATE TABLE IF NOT EXISTS Transaction (TransactionId INT, AccountNo STRING, Time STRING, Location STRING, Amount FLOAT, Alert STRING)");
```

Account & Transaction HQL

```
public Account getAccount(String accNo) {
    Dataset<Row> account = spark
            .sql("select * from Account where AccountNo='" + accNo + "'");

    Iterator<Row> rs= account.toLocalIterator();
    Account result = null;
    if(rs.hasNext()){
        Row arg0 = rs.next();
        String accNo1 = arg0.getString(arg0.fieldIndex("AccountNo"));
        String user = arg0.getString(arg0.fieldIndex("User"));
        String homeAddress = arg0.getString(arg0.fieldIndex("HomeAddress"));
        String accountType = arg0.getString(arg0.fieldIndex("AccountType"));
        result = new Account(accNo1, user, homeAddress, accountType);
    }
    return result;
}

// Get the most 24 transactions of User based on Account NO
public List<Transaction> getRecentTransactions(String AccountNo) {
    Dataset<Row> transactions = spark
            .sql("select * "
            + "from Transaction " + "where Alert <> 'Fraud' and AccountNo = '"
            + AccountNo + "' " + "Limit 24");
    Iterator<Row> rs= transactions.toLocalIterator();
    List<Transaction> result=  new ArrayList<Transaction>();
    while(rs.hasNext()){
        Row arg0 = rs.next();
        int transId = arg0.getInt(arg0.fieldIndex("TransactionId"));
        String accNo = arg0.getString(arg0.fieldIndex("AccountNo"));
        String transactionTime = arg0.getString(arg0.fieldIndex("Time"));
        String location = arg0.getString(arg0.fieldIndex("Location"));
        float amount = arg0.getFloat(arg0.fieldIndex("Amount"));
        String alert = arg0.getString(arg0.fieldIndex("Alert"));
        result.add(new Transaction(transId, accNo, transactionTime,
                location, amount, alert));
    }
    return result;
}
```

Insert New Transaction HQL

```
public void insertNewTransaction(JavaRDD<Transaction> rdd) throws AnalysisException{
    Dataset<Row> record = spark.createDataFrame(rdd, Transaction.class);

    record.createOrReplaceTempView("currentTrans");

    Dataset<Row> df = spark
    .sql("select transactionid, accountno, time, location, amount, alert from currentTrans");
    df.show();
    df
     .write()
     .format("orc")
     .mode(SaveMode.Append)
     .insertInto("default.transaction");
}
```

Transaction Fraud Detection

- Simple algorithm to compute " Transaction" points – Based on
  - Location of Transaction
    - Same City, State, Country -> 0 point
    - Same State and Country -> 5 points
    - Different Country - > 11 points
  - At most 24 of Recent Transactions
    - Existed Location -> 0 point
    - New Location
      - Same Country -> 5 points
      - Different Country -> 11 points
  - Average Amount of 24 Recent Transactions
    - <= (Avg + $500) -> 0 points
    - > (Avg + $500) -> 5 points

➔ **ENRICHED (<=10 points)** otherwise **FRAUD (>10 points)**

Demo