Hojin Ryoo

M&T Bank

David Eyerly

May 22nd, 2019

Bank Skeleton Dataset Project

Introduction:

Our goal was to create a dataset in Neo4j that could be used in a classroom or competition environment to give students the opportunity to explore real world problems. The reason why we chose Neo4j was because of its Graph Database structure, which lends itself very well to a visual learning style.

This dataset deals with bank fraud detection. Students will use queries to try to find the correct patterns that correlate with bank fraud scenarios. While the dataset currently explores only a number of scenarios, the dataset can be edited to encompass other types. This dataset is therefore a flexible tool for any individual or group who wishes to make their own scenarios.

Background:

Specifically, this dataset deals with three kinds of money laundering. First, **concentration**, which is when money is collected/received from multiple accounts by individuals and transmitted usually in form of wire to an overseas destination on a regular/scheduled basis. Second, **large cash deposits**, when an individual deposits cash just under $10,000 over at least a three day period. Third, **velocity** when cash is transacted rapidly over a short period of time.

The dataset contains thousands of nodes and relationships. In Neo4j nodes are entities containing attributes and relationships connect nodes together. Students should be expected to be able to query through these nodes and relationship to find patterns that correlate to money laundering schemes.

Student Objectives:

* Identify Money Laundering Scenarios
* Identify Parties (a.k.a. Customers) and Accounts involved and how much money is flowing between them
* Explain/Document the process in which queries were created
* Be able to query through high volume and/or high variable dataset

Analysis and Design:

The data was generated through **Mockaroo**, which is an open source web app that ise used to generate large scale datasets. We first used actual bank records as a template to figure out what data would be relevant for detecting money laundering scenarios. The bank record data was simplified and copied into Mockaroo’s data generation schema, where we generated CSV files.

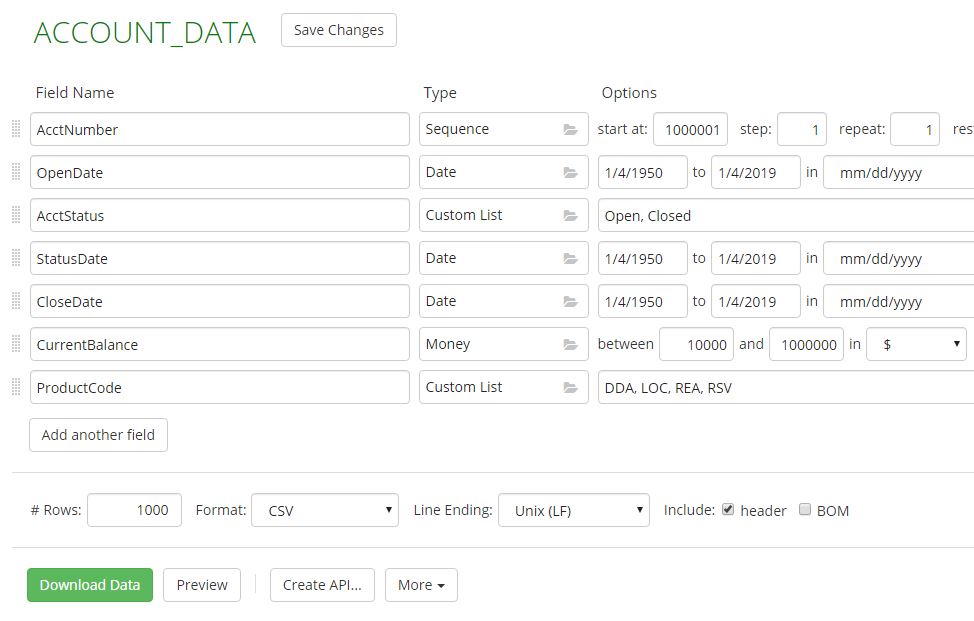


Figure 1 Mockaroo Account Schema

Next, we decided on three main subsets of data; **Account**, **Transaction**, and **Customer**. Individual and company customers were generated separately in Mockaroo. After examining the different attributes of these three subsets of data, we concluded that the **Transaction Date** subset could be derived from Transaction. In Neo4j, these subsets will be translated into nodes. Date being a node made it easier to identify the **velocity** scenario, as it allows for querying through the relationships between Date, Transaction, and Account We also derived three more subsets of data to generate relationships within Neo4j. First **Customer to Account**, second **Transaction to Date**, and third **Transaction to Account**. These subsets of data are generated in Mockaroo, and then imported into Neo4j to be queried for patterns.

Implementation:

We initially encountered a problem, where we wanted these CSV files to be accessible and easy to edit. Microsoft Excel is a great CSV editing tool, but unfortunately when converting CSV files to Excel files leading zeroes in data entries are lost. To fix this error, the CSV files were renamed to TXT files and imported directly into Excel. Through this method, leading zeroes were kept throughout the data.

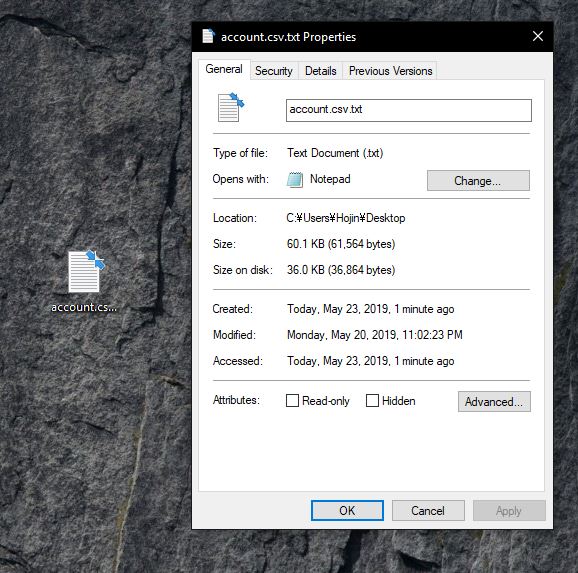


Figure 2 CSV file renamed to txt file

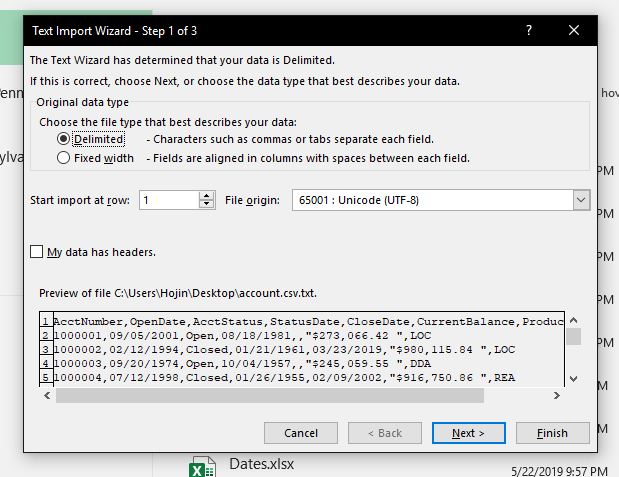


Figure 3 Import Step 1

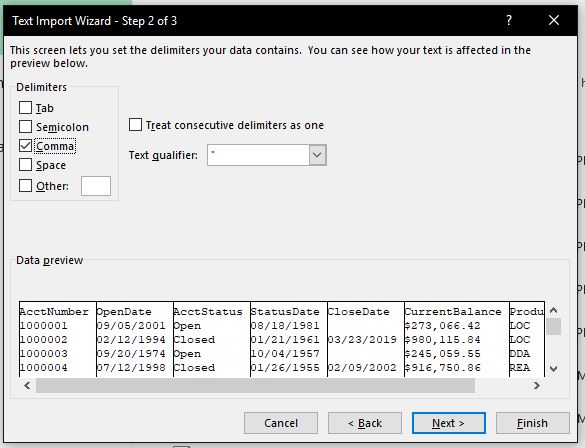


Figure 4 Import Step 2 Setting the column delimiters to commas

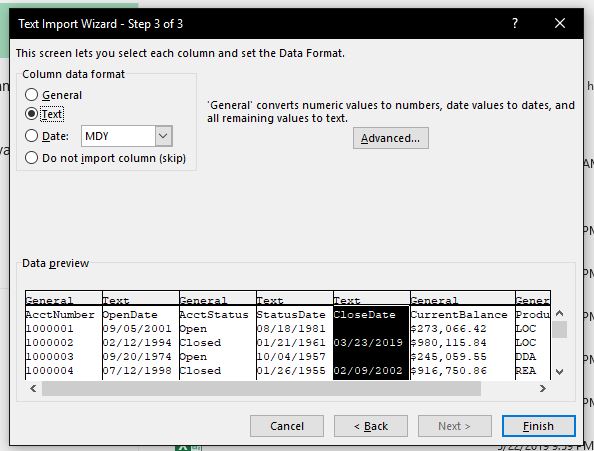


Figure 5 Import Step 3 Reformatting leading zero columns to text format instead of general format

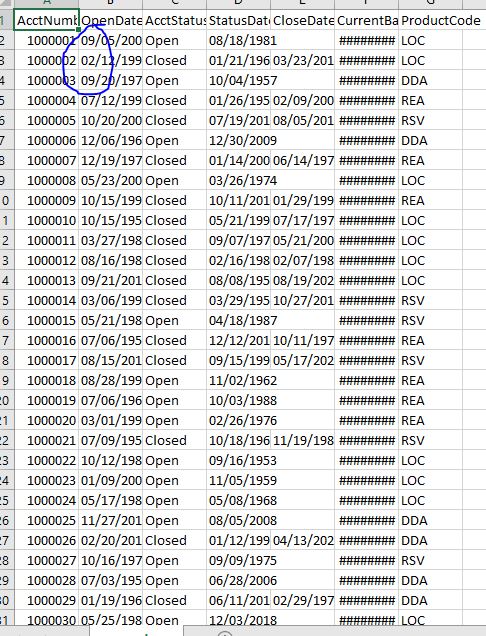


Figure 6 Leading zeroes now are present

Next, we needed to insert the scenarios into the excel files. First, the accounts committing the bank fraud were created and compiled into a “Cheat Sheet”, so that the patterns could be double checked once inserted. Their corresponding customer ID’s were also compiled, so that it is easier to doublecheck individual names and company names. When changing the accounts for the scenarios, I would recommend doing the same thing so that the answers and their corresponding relevant data are all in one place for accessibility. In the case of individual customers for this dataset, (account numbers below 167) the ending digits of the account ID are the same as the customer ID.

Individual fields were edited within the datasets to create the scenarios. For the concentration scenario, the account numbers were changed in the transaction dataset so that multiple accounts would be funneling transactions to a single account. The customer dataset was edited so that those the corresponding customers would have origins outside of the US. Some of the names were also changed for fun, to better reflect their criminality int this exercise. (Ex. Dastardly Dude Devin)

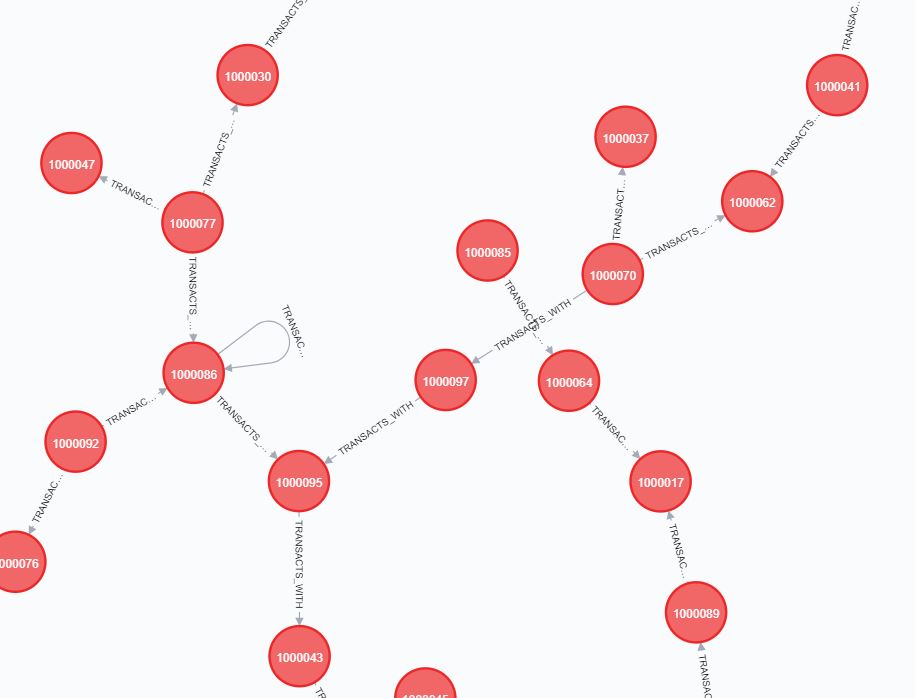
From there, I exported the Excel files as CSV files and imported them into Neo4j. In to import them, I had to move the CSV files to a local location within the Neo4j database directory. Once the nodes were imported, I also had to create indexes for some of the attributes to act as primary keys for the relationships to compile faster. Here are some examples of importing them as nodes and relationships using the cipher database language, and their visual representation in Neo4j.

Importing Account Nodes using a CSV file:

USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:///account.csv" AS row

CREATE (:Account {accountNum: row.AcctNumber, openDate: row.OpenDate, accountStatus: row.AcctStatus, statusDate: row.StatusDate, closeDate: row.CloseDate, currentBalance: row.CurrentBalance, productCode: row.ProductCode});



Importing the Transaction to Date Relationship from a CSV:

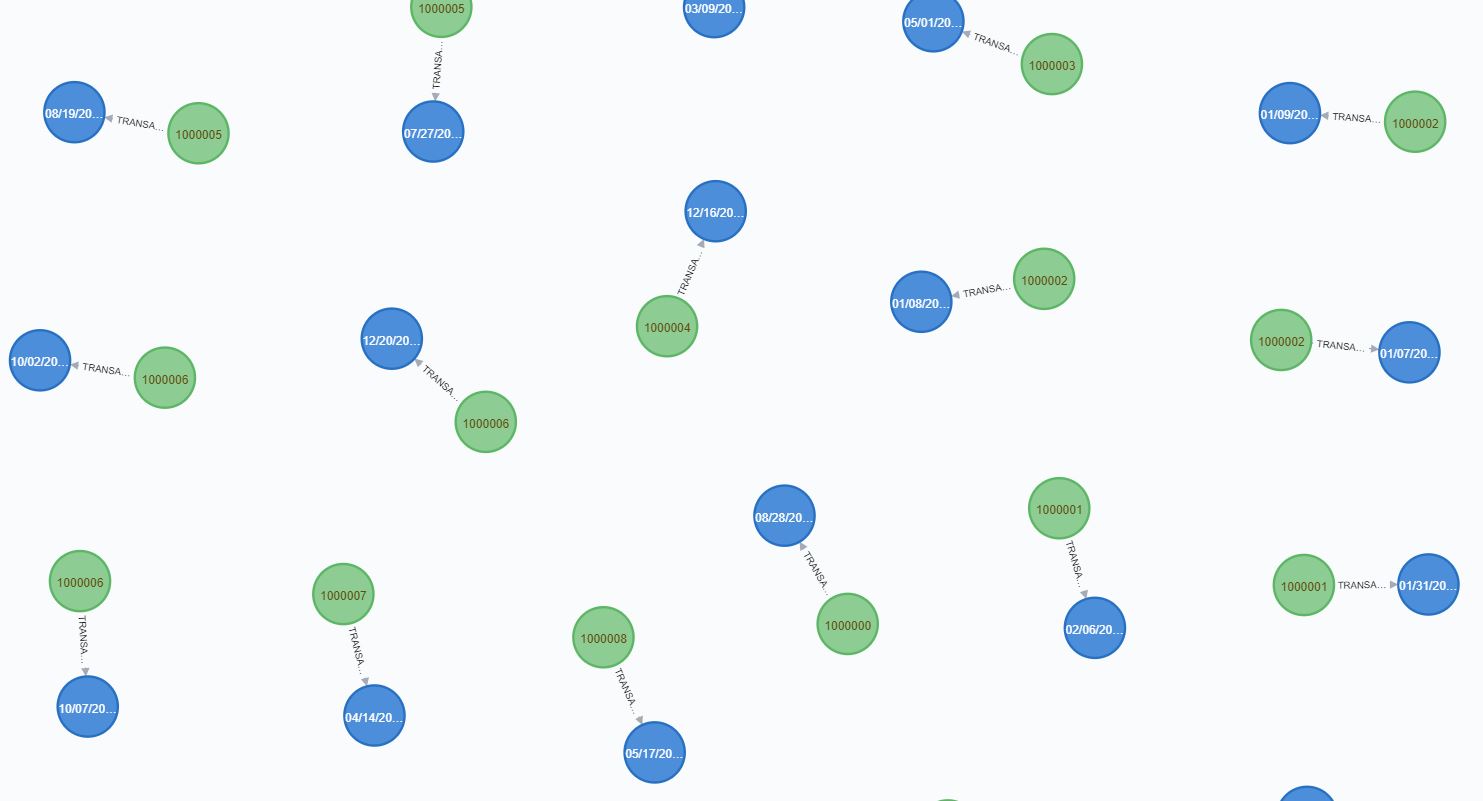
USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:///transaction\_to\_date.csv" AS row

MATCH(transaction:Transaction {tracewireID: row.Trace\_Wire\_ID})

MATCH(date:Date {date: row.Transaction\_Date})

MERGE(transaction)-[:TRANSACTION\_DATE]->(date);

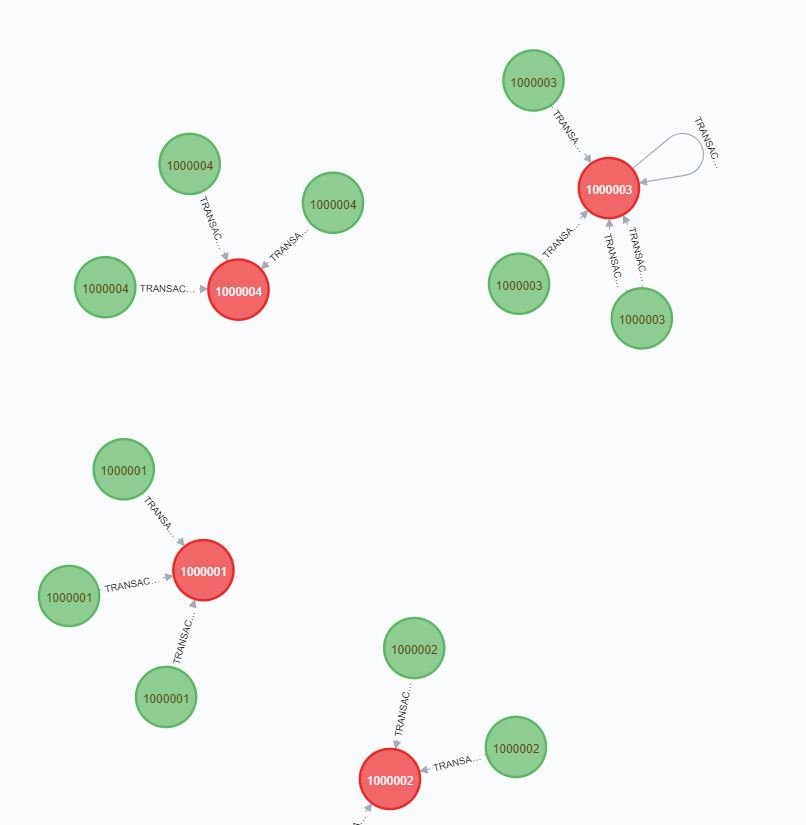


Creating Transaction to Account Relationship:

MATCH(a:Account),(t:Transaction)

WHERE a.accountNum = t.tAccount1

MERGE(t)-[:TRANSACTION\_ACCOUNT\_1]->(a);

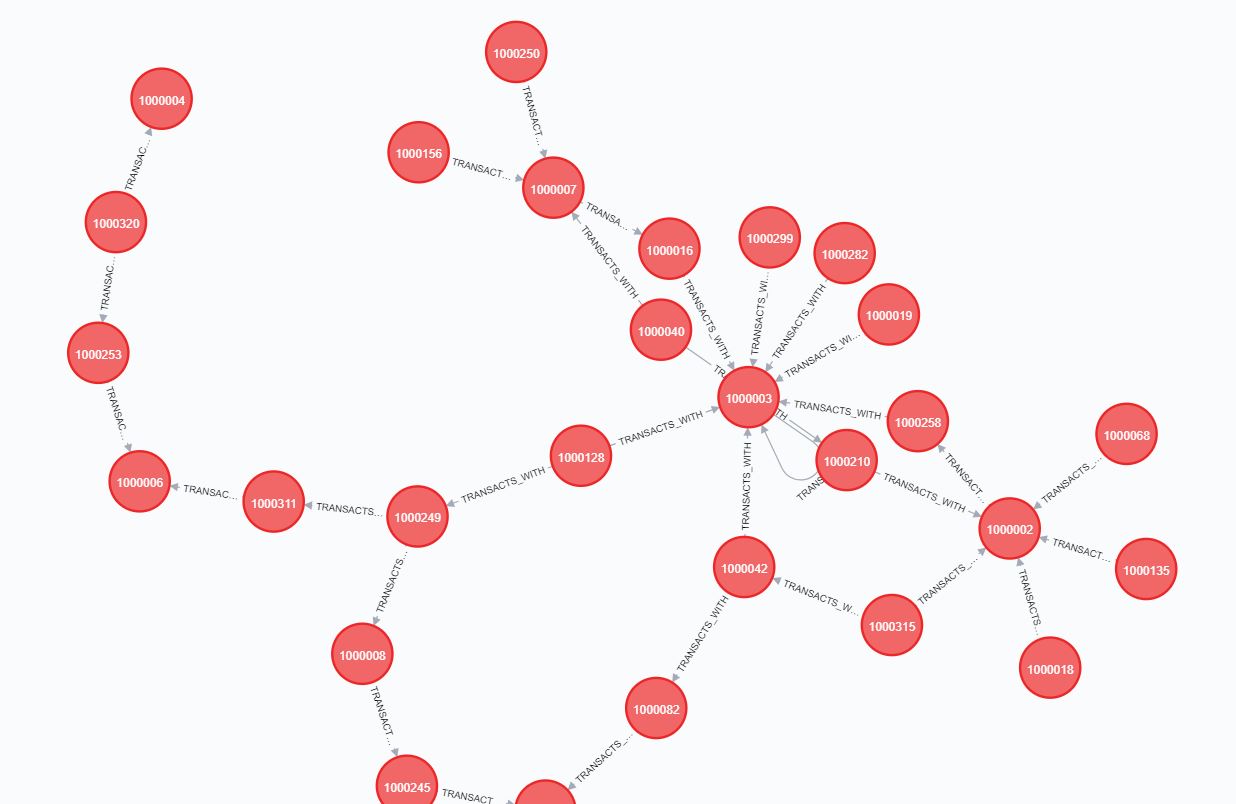


Creating Account to Account Relationship (Transactions between Accounts):

MATCH(a1:Account),(t:Transaction),(a2:Account)

WHERE a1.accountNum = t.tAccount1 AND a2.accountNum = t.tAccount2

MERGE(a1)-[p:TRANSACTS\_WITH]->(a2);



Testing and Results:

The testing and results section are combined, as they are both connected in our dataset. Testing whether the relationships and nodes are accurate, and whether they can be queried through reasonably will essentially be our results.

We had already done many test runs of importing the CSV files into Neo4j, and due to this fact there were many duplicate files running around. There were many times where in the finished run the files were imported correctly, but the datasets themselves did not carry the scenarios, or they were formatted incorrectly and would produce the incorrect nodes, their properties, and relationships. Thankfully after carefully combing through each dataset using the cheat sheet we created in the beginning, the CSV files imported correctly, and all the nodes and relationships were correctly corresponding to one another. At a glance, you may be able to notice some patterns already just from the initial screenshots of the relationships.

The last and final test was to determine whether it was possible to query the information to achieve the student objectives. For concentration, an easy query to do is search for accounts that have a count of 5 or more accounts that it has a relationship with. An example of such a query is displayed below.

All of the queries were able to find multiple instances of the money laundering scenarios. This also means that the scenarios are written into the datasets such that it is possible to query over multiple relationships to find specific property/attribute values. This is significant, as you could use just one query to find a transaction ID (tracewireID), the amount of money in the transaction, the account number, then name of the customer, and where he lives.

FINDING CONCENTRATIONS

Query

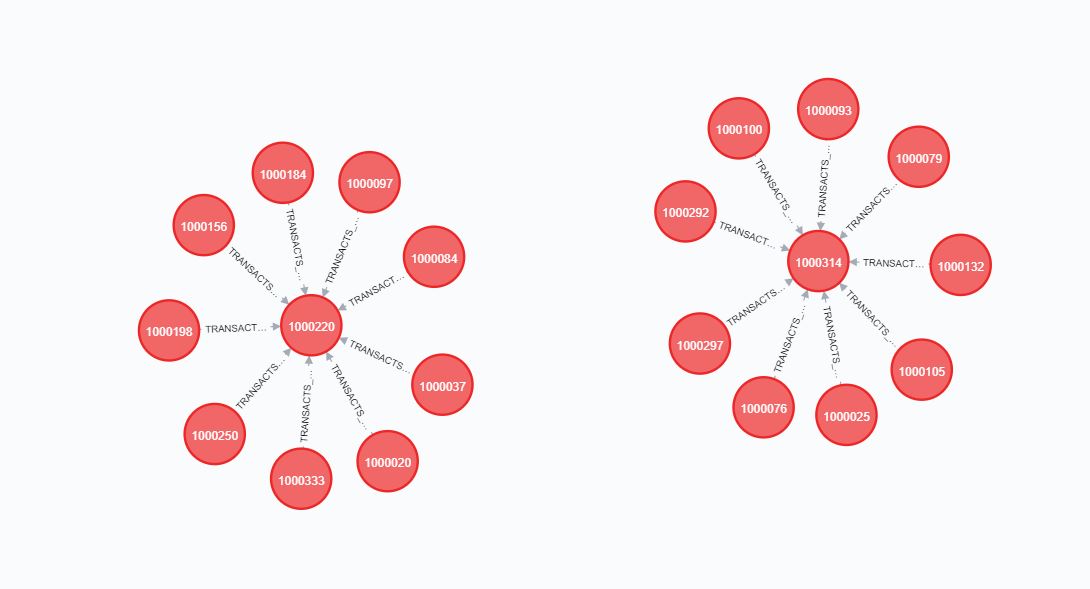
MATCH (n:Account)<-[:TRANSACTS\_WITH]-(p:Account)

WITH n as Account2Nodes, size(()-[:TRANSACTS\_WITH]->(n)) AS count, p as pika

WHERE count > 8

RETURN DISTINCT pika, Account2Nodes, count

What this query does, is it queries for every single TRANSACTS\_WITH relationship. Then, I use the size() scalar function to find accounts that have many transactions that are being funneled into it by other accounts. It then returns distinct accounts that are funneling the money, the number of transactions, and the accounts being funneled into.



FINDING LARGE CASH DEPOSITS:

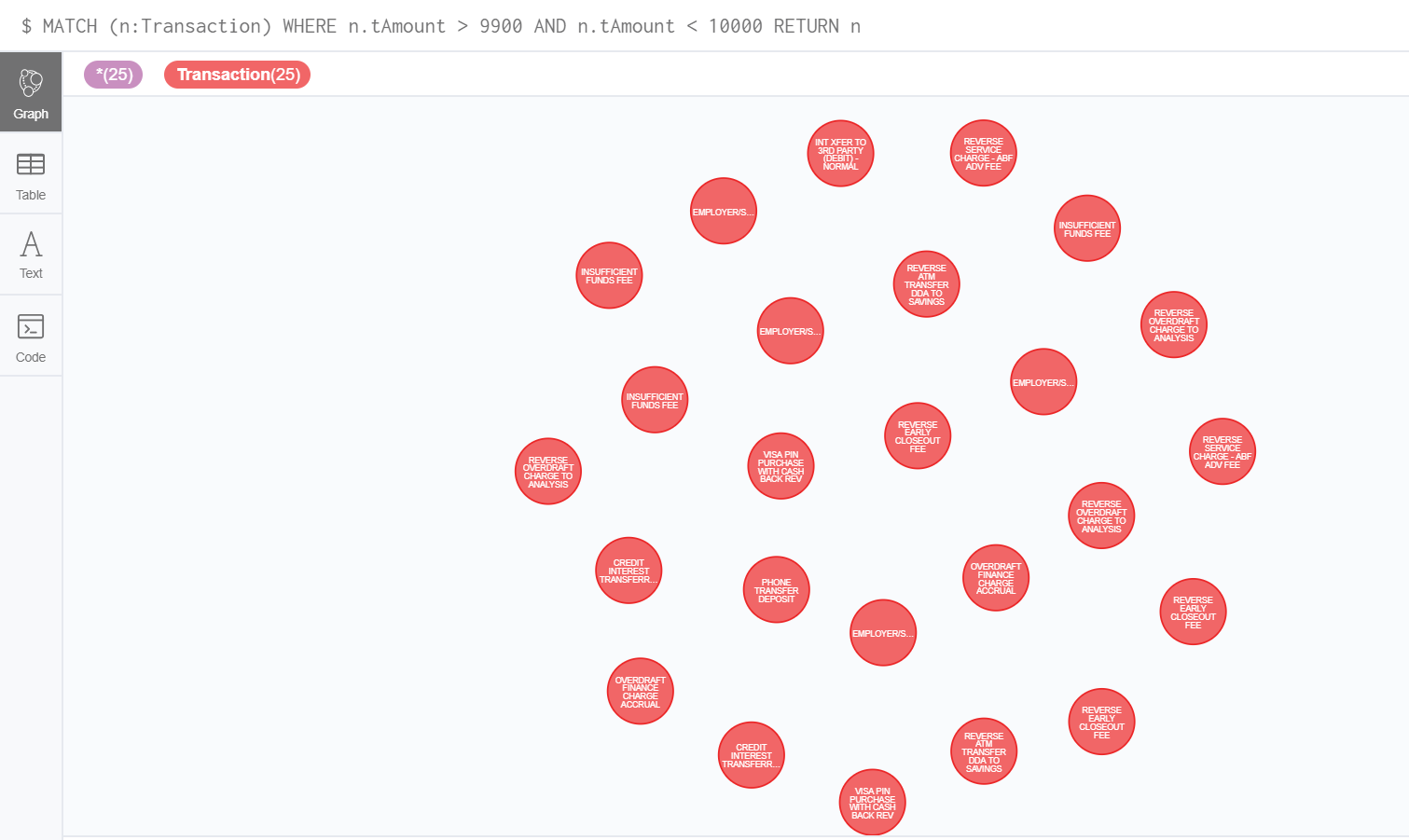
Query

MATCH (n:Transaction)

WHERE n.tAmount > 9900 AND n.tAmount < 10000

RETURN n;

This query searches for Transactions where the transaction amount is between 9900 and 10000 dollars.



FINDING VELOCITY:

Query

MATCH (p:Transaction)-[:TRANSACTION\_DATE]->(m:Date)

RETURN p.tAccount1, m.date

ORDER BY p.tAccount1, m.date

This query searches for the account numbers and dates of all transactions, and then orders them by account number and then date after. This makes it really easy to spot where an account has transactions that are occurring over a small amount of time, at least within a small dataset such as this one. If we were to scale up to a larger dataset, this method would be very difficult to spot these sorts of transactions.



Conclusion:

As I have personally never used Neo4j or had any experience with the Cypher programming language before starting this project, I learned a lot over the course of this year creating this dataset. Learning these materials on my own translated very well into school, as it introduced me to topics and concepts that came up many times during my database classes at Penn State University. Collaborating with David Eyerly and being able to talk with Balaji was also a very valuable experience. I was able to work on my networking and scheduling skills alongside gaining experience with databases and database management tools.

Neo4j is a great tool for graphing bank fraud scenarios, and this particular dataset that we’ve created scales up very well too. Using Mockaroo the dataset could easily be scaled up to tens of thousands of data entries for each subset of data, such as accounts, transactions, and customers. The dataset currently explores only a number of scenarios, but the dataset can also be edited to encompass others types by directly editing the excel files and then exporting them as new CSV files. This dataset is therefore a flexible tool for any individual or group who wishes to make their own scenarios.

Overall, the project went very well. I wouldn’t mind continuing working on the dataset in my spare time, as I’m sure there are other things that could be added, and more efficient ways of querying the data that I haven’t thought of yet. There are a lot of tools in Cypher to structure graph databases, and I’m sure that there are better ways of importing CSV files to structure them in interesting ways.