# Credit Card Fraud Detection

Introduction

As per a Fortune 500 financial Service Company case study, 2.2 million financial transactions are conducted per month. Majority of these transactions are processed by the company's automated fraud detection system and few of them are submitted for manual review before the transfer of money is completed. Traditionally, these manual review were conducted in relational database that would require carrying out a series of complex joins on different tables to model a graph showing such relations. These are queries are expensive to run and results into poor performance as the data size of such a graph increases.

Neo4j can help model and monitor data about customers, accounts, devices, locations and other such information to detect transactions in the graph.

Problem Statement

We intent to detect fruadulent transaction based on the disputed status of the credit card transactions in our data set using Neo4J.

Dataset

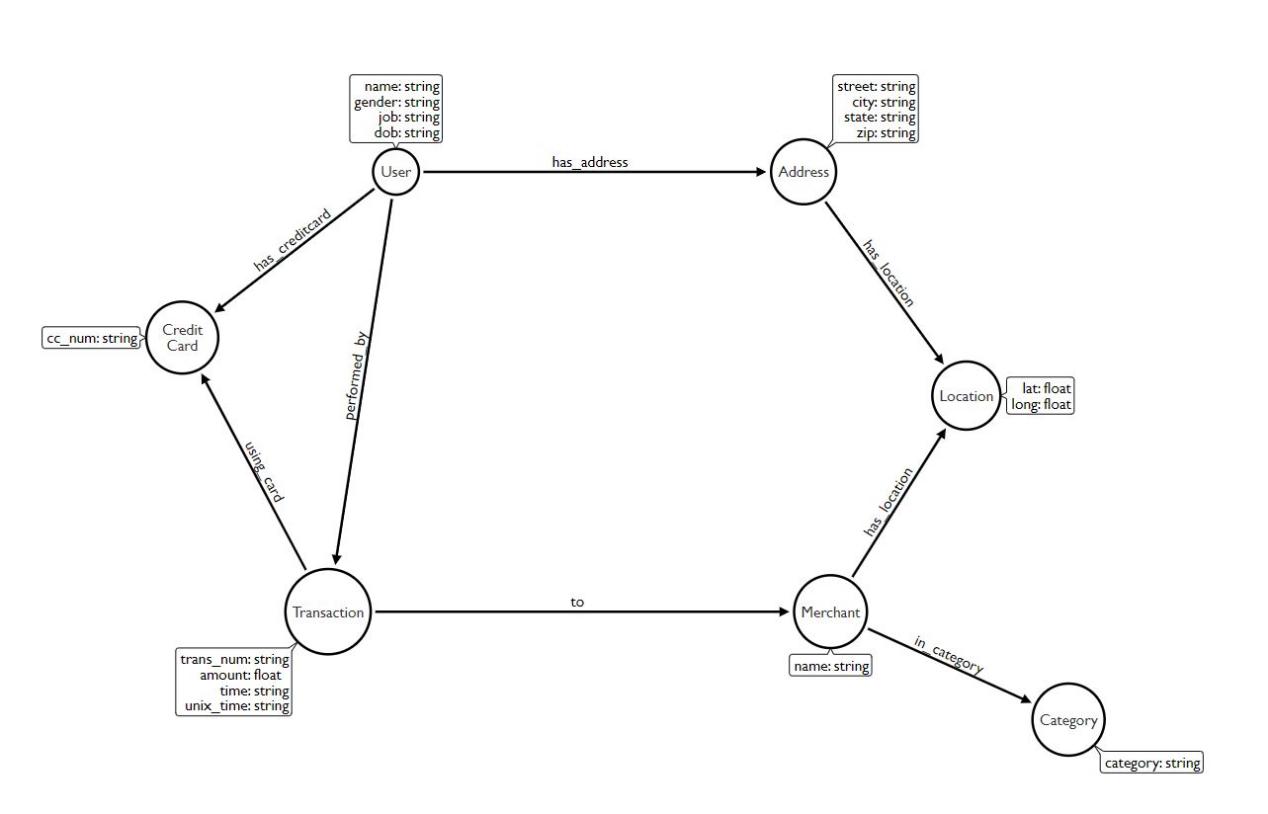
As credit card transaction are private and cannot be obtained, we will be using synthetic data obtained from kaggle - Credit Card Fraud Detection Dataset - from the following link:

<https://www.kaggle.com/datasets/kartik2112/fraud-detection>

It contains simulated credit card transactions containing transactions from January 1st 2019 to 31st December 2020 with 1000 customers and 800 customers. It contains information such as Transaction Number, Credit card number, user information, customer address, merchant address, disputed status etc.

Data Model

Using Arrow Tool, we develop our data model which is as seen below:



Our projects contains the following nodes:

1. User: They perform transactions to several different merchants. The node contains the user’s information such as name, gender, job and date of birth.
2. Credit Card: It is owned by the user and used to perform the transaction. The node contains the credit card number.
3. Transaction: It contains the transaction data such as transaction number, amount, time, unix\_time and is\_disputed.
4. Merchant: Transactions are done by the users to the merchants. This node contains the merchant’s name.
5. Category: Each Merchant belongs to a category. This node contains the category name.

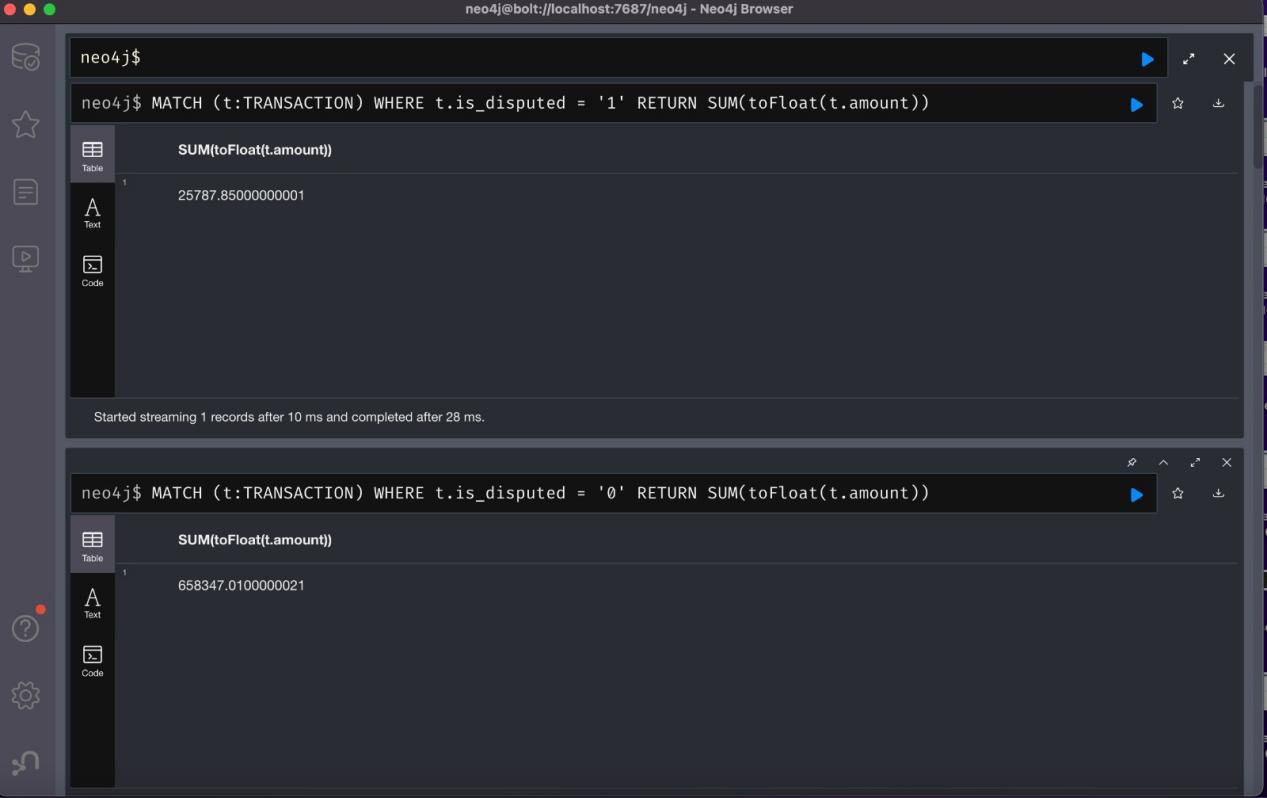
Data Pre-processing

Data cleaning and transformation was performed on Google Colaboratory where the following tasks were performed:

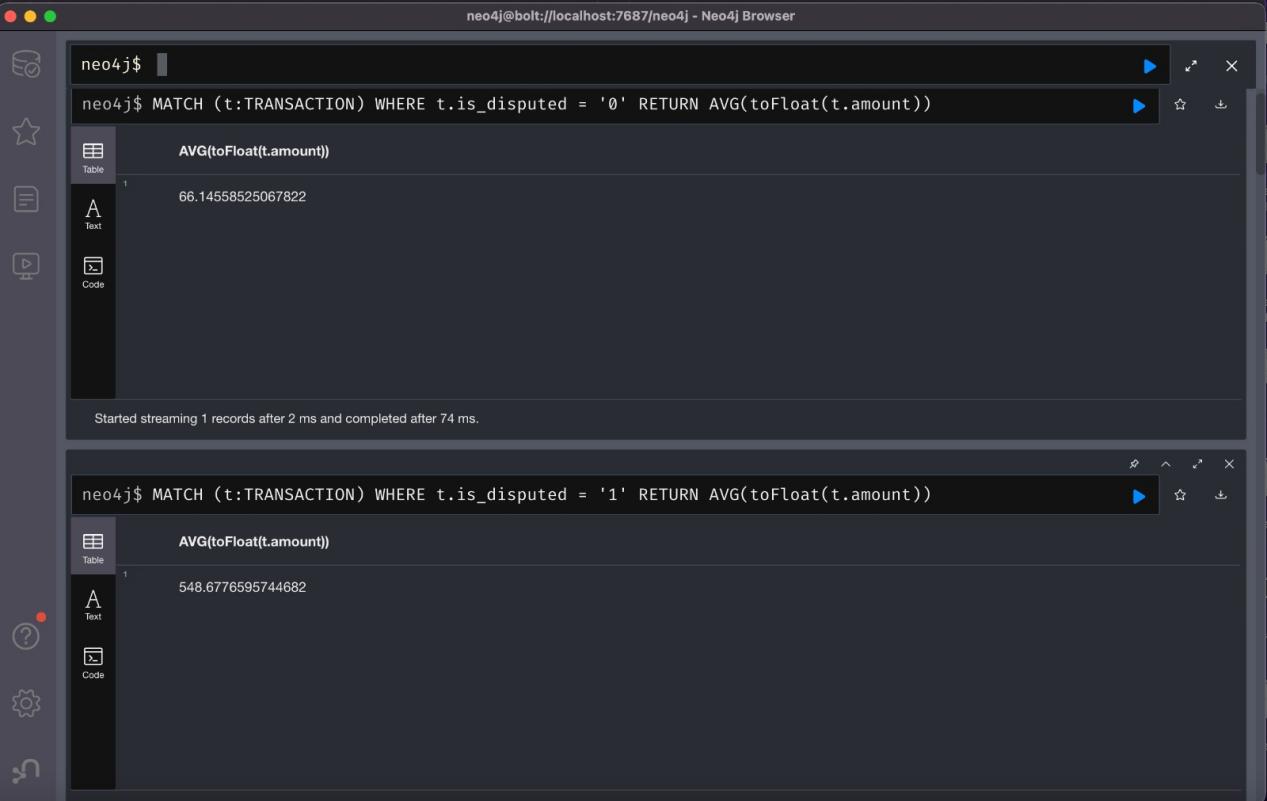
1. Null and Nan values were removed
2. Duplicate Entries were removed

Aggregation Operations

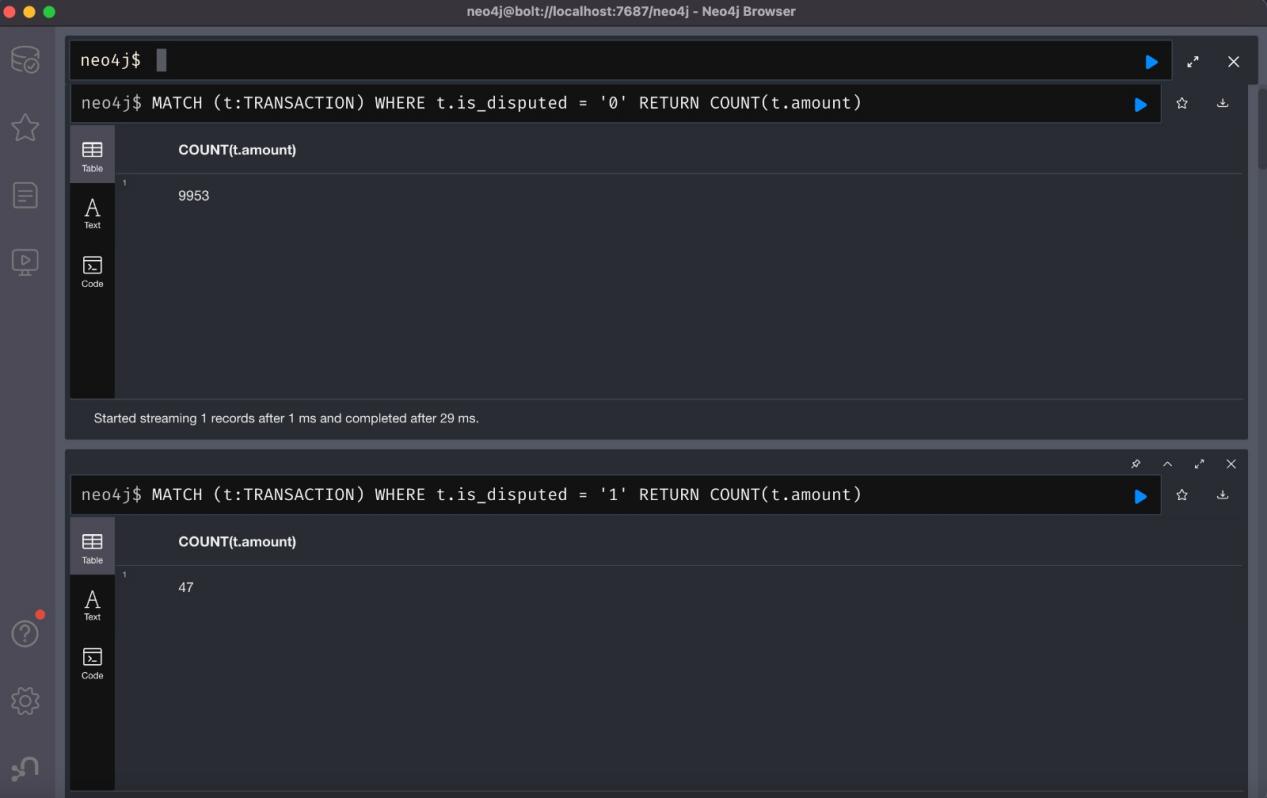
1. Sum of the amount of all legitimate and fraudulent transactions



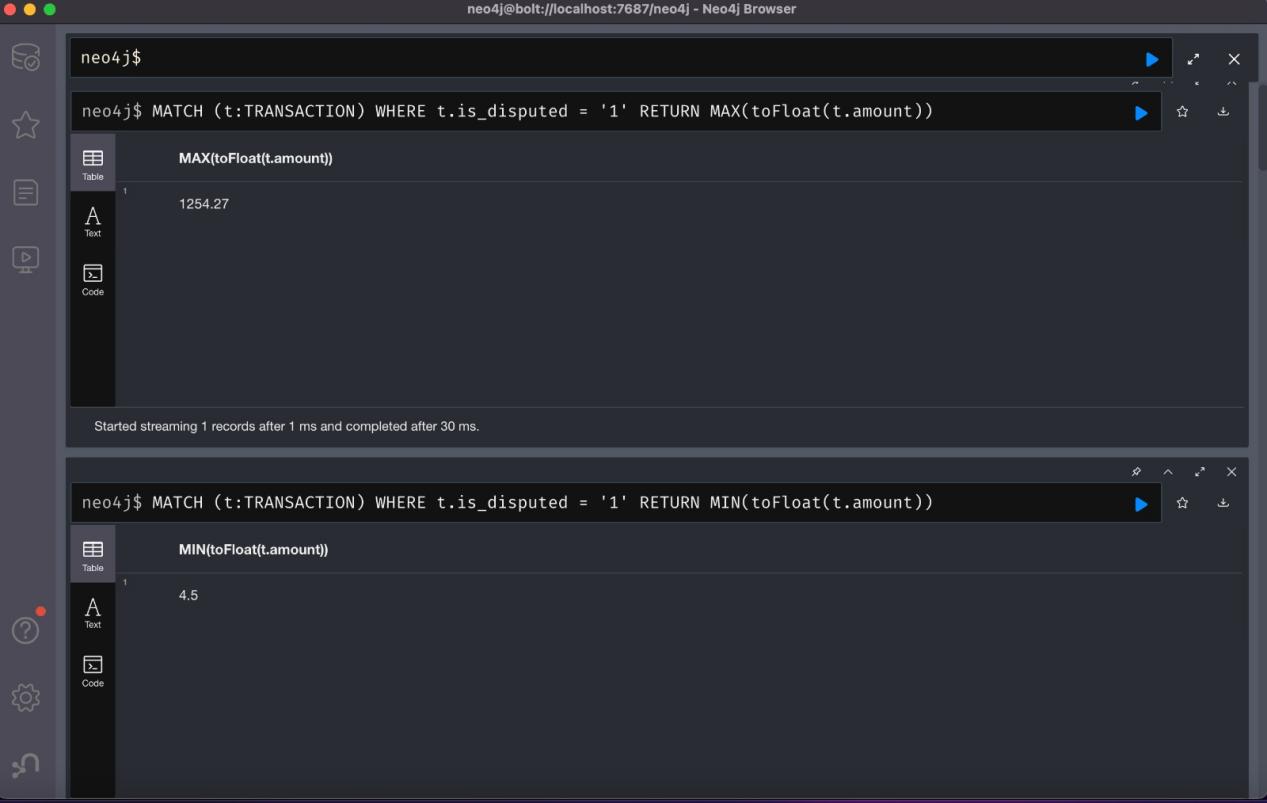
1. Average amount of all legitimate and fraudulent transactions



1. Count of all legitimate and fraudulent transactions



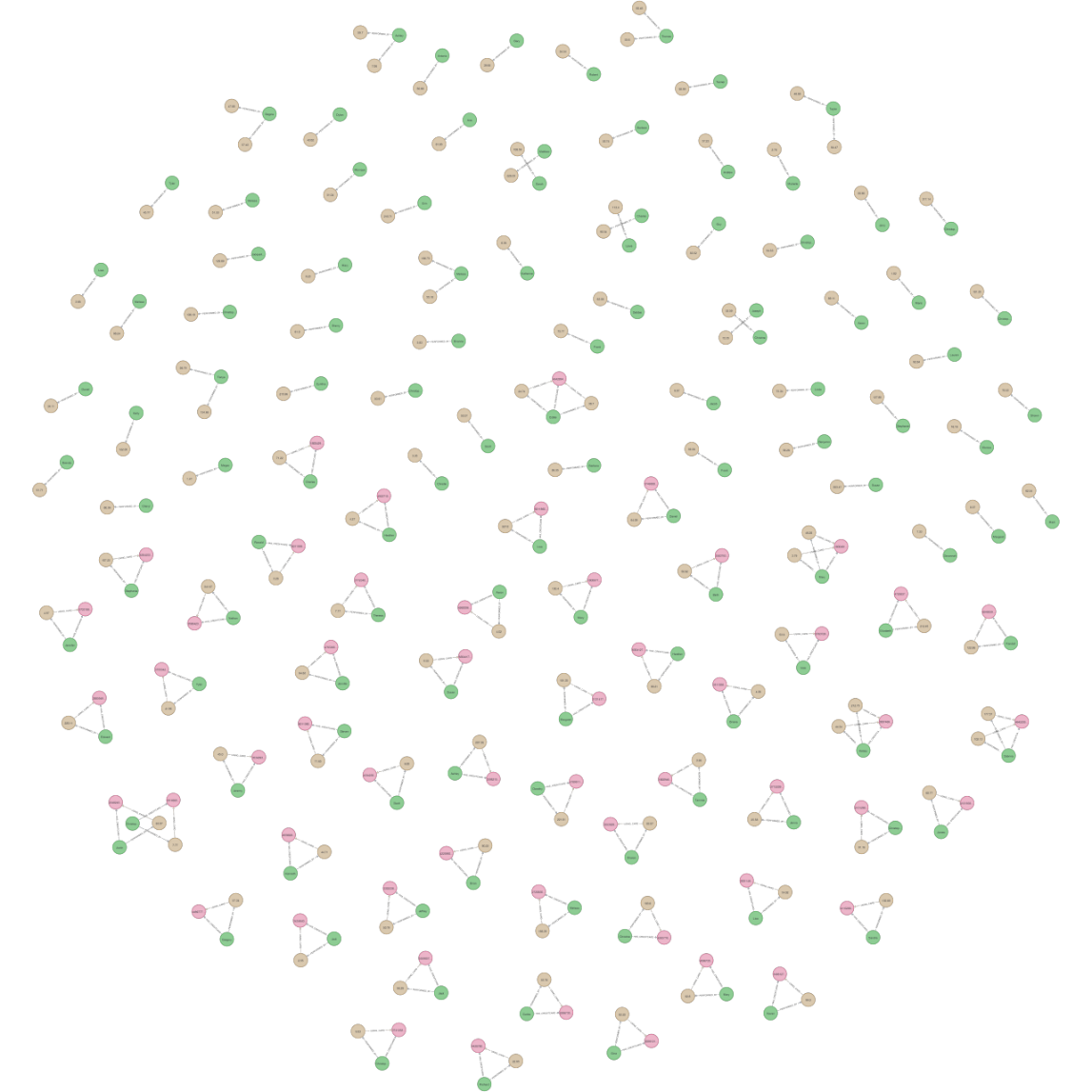
1. Maximum and minimun of the amount fraudulent transactions



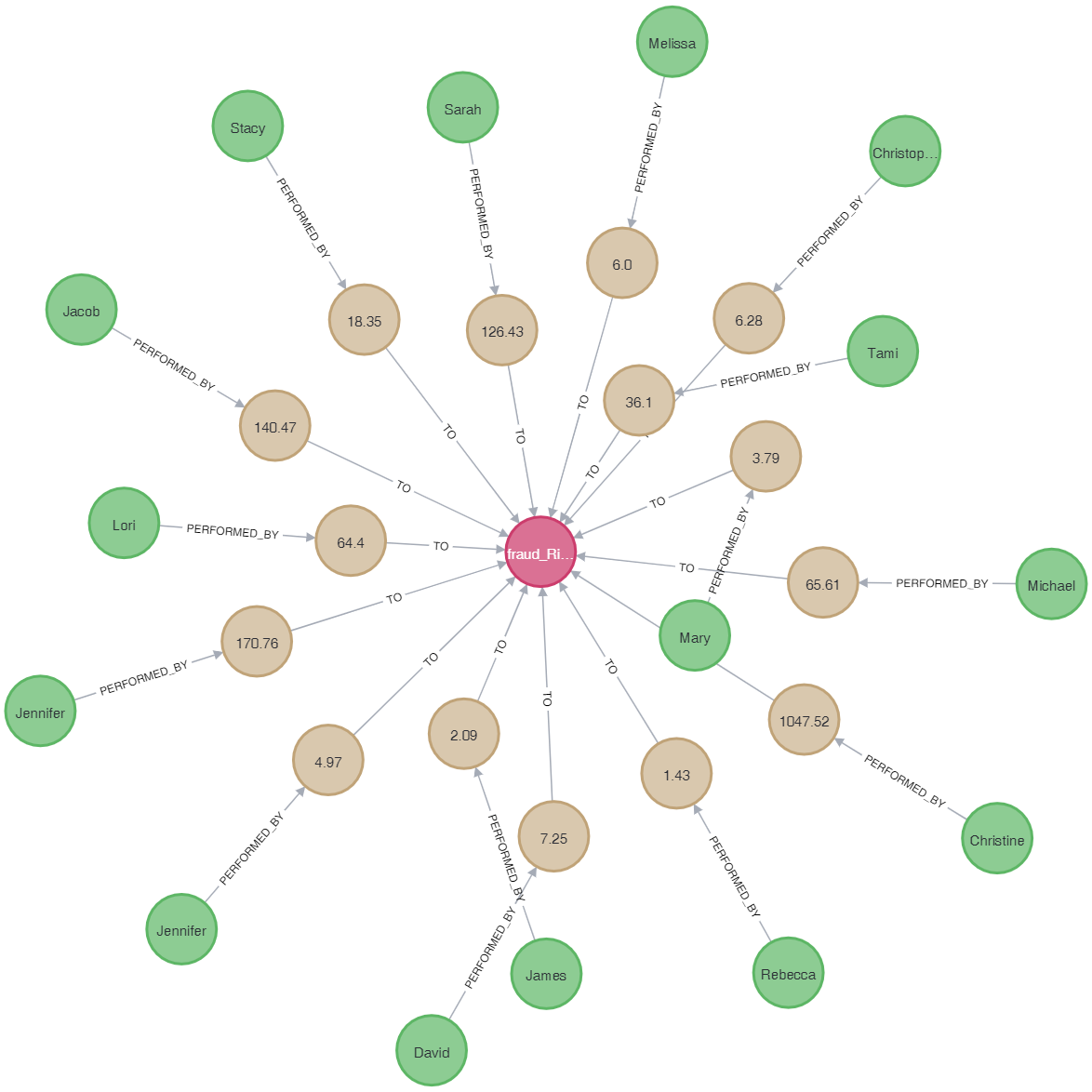
Database Queries

1. Get all nodes

MATCH (all) RETURN all;

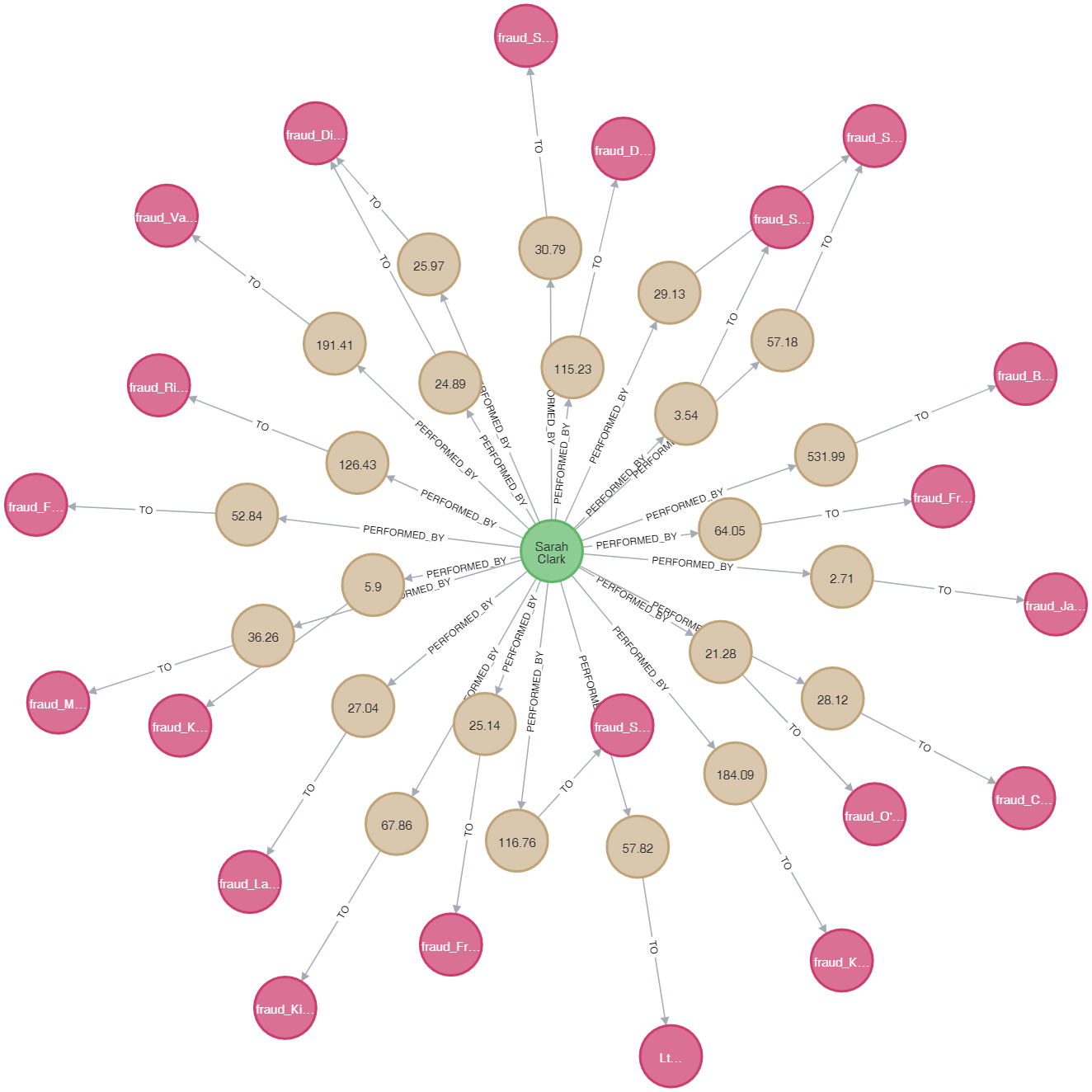


1. MATCH (u:USER) -[:PERFORMED\_BY]-> (t:TRANSACTION) -[:TO]-> (m:MERCHANT) RETURN u, t, m LIMIT 15;



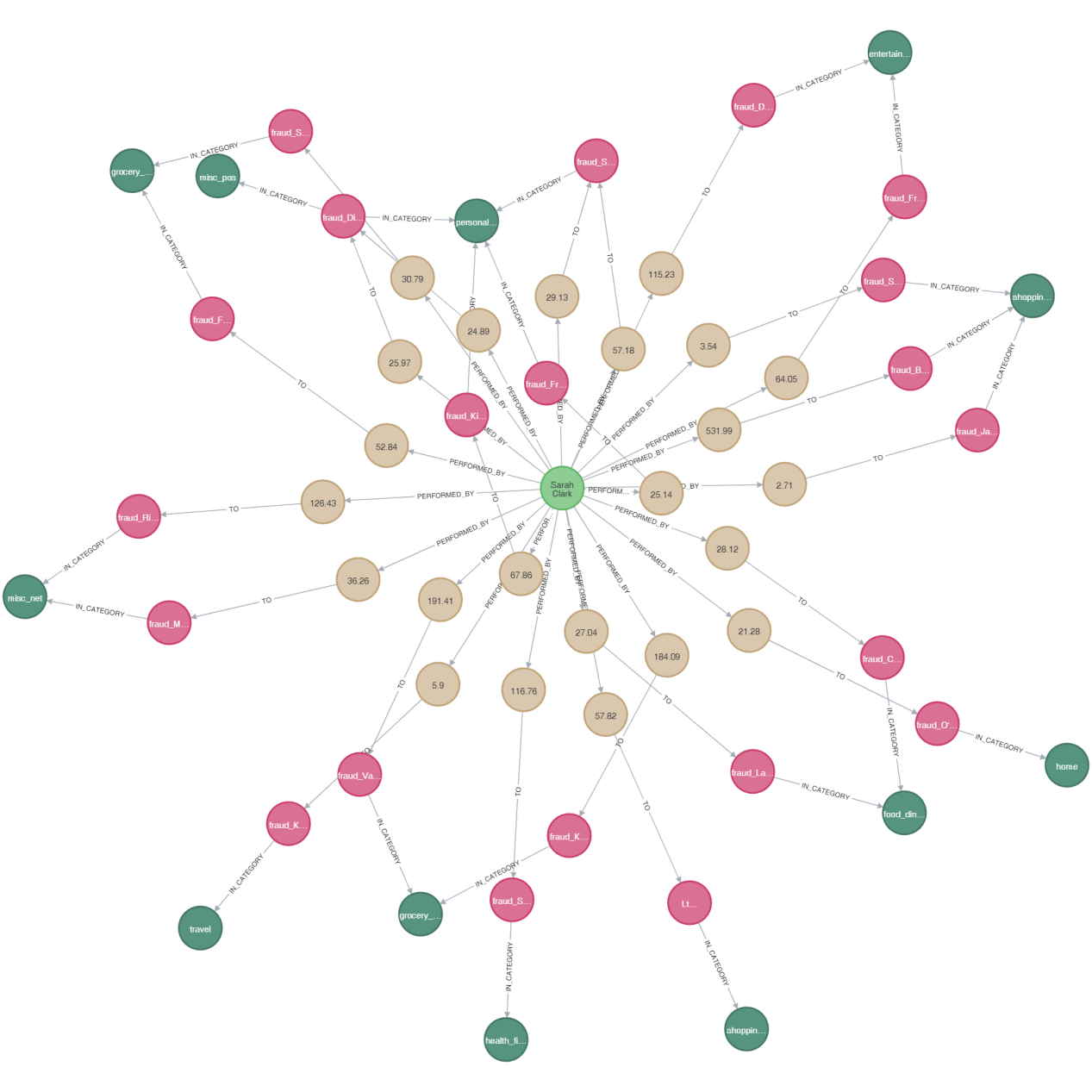
1. Get all transactions performed by a user:

MATCH (u:USER {name: 'Sarah Clark'}) -[:PERFORMED\_BY]-> (t:TRANSACTION) -[:TO]-> (m:MERCHANT) RETURN u, t, m;



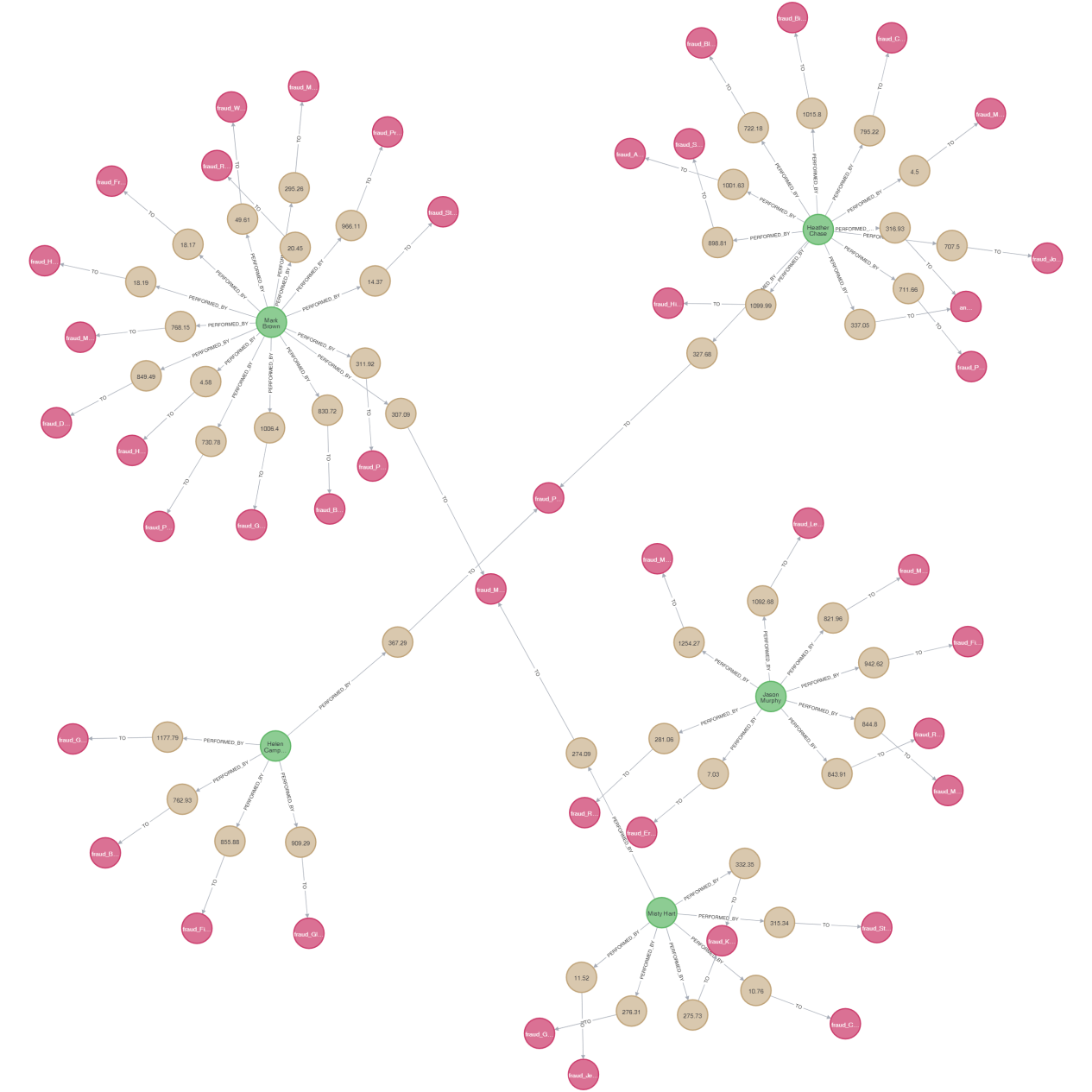
1. Get Transactions and category of merchant

MATCH (u:USER {name: 'Sarah Clark'}) -[:PERFORMED\_BY]-> (t:TRANSACTION) -[:TO]-> (m:MERCHANT) -[:IN\_CATEGORY]-> (c:CATEGORY) RETURN u, t, m,c;



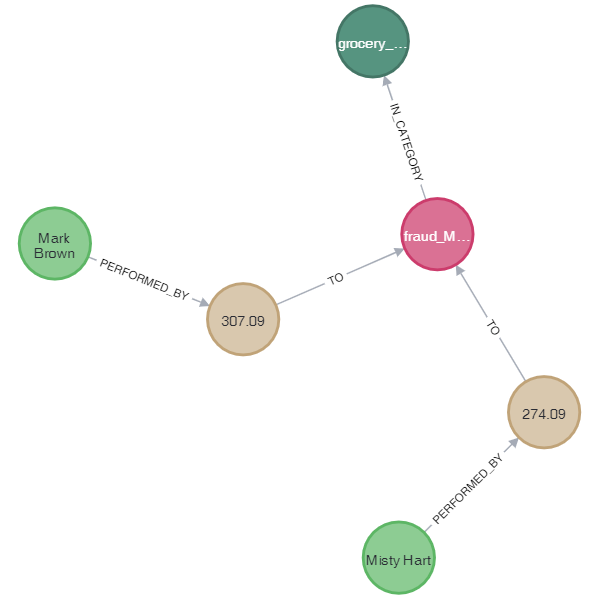
1. Get all Disputed Transactions

MATCH (u:USER) -[:PERFORMED\_BY]-> (t:TRANSACTION) -[:TO]-> (m:MERCHANT) WHERE t.is\_disputed = '1' RETURN u,t,m;



1. Merchants that have fraudulent Transactions

MATCH (u:USER) -[:PERFORMED\_BY]-> (t:TRANSACTION) -[:TO]-> (m:MERCHANT {name: "fraud\_Moen, Reinger and Murphy"}) -[:IN\_CATEGORY]-> (c:CATEGORY) WHERE t.is\_disputed = '1' RETURN u, t,m, c;



1. **Customer and Merchants involved in a disputed case**

MATCH (u:USER) -[:PERFORMED\_BY]-> (t1:TRANSACTION) -[:TO]-> (m1:MERCHANT)

WHERE t1.is\_disputed = '1'

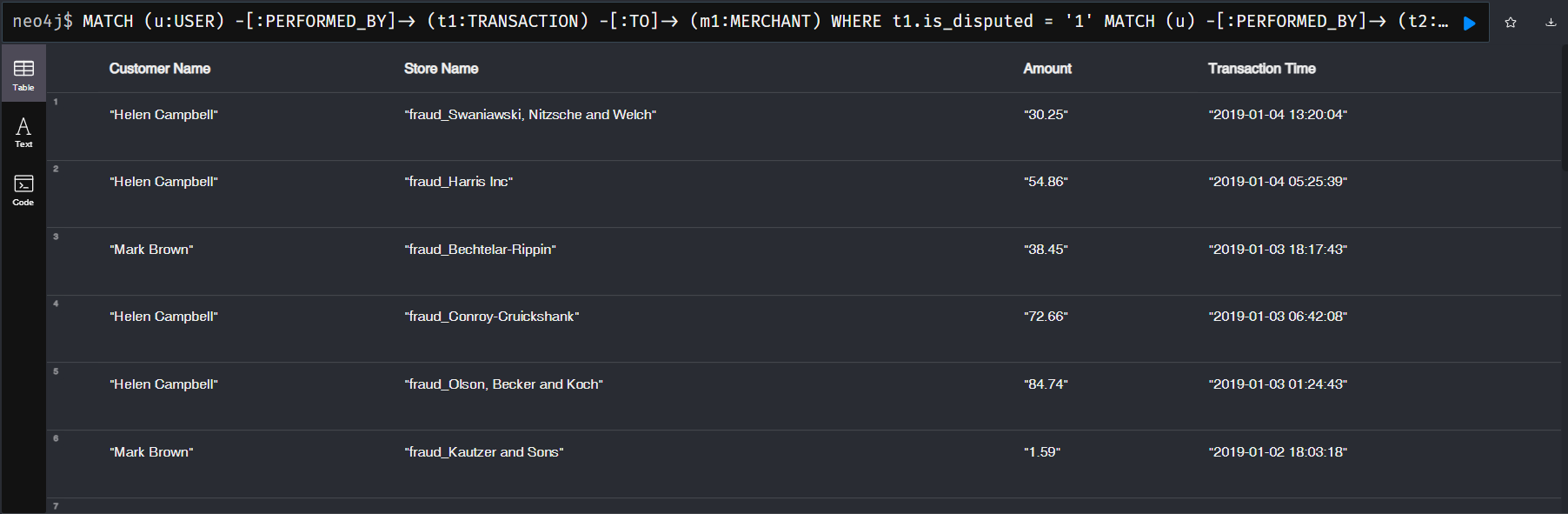
MATCH (u) -[:PERFORMED\_BY]-> (t2:TRANSACTION) -[:TO]-> (m2:MERCHANT)

WHERE t2.is\_disputed = '0' AND t2.time < t1.time

WITH u, m2, t2 ORDER BY t2.time DESC

RETURN DISTINCT u.name AS `Customer Name`, m2.name AS `Store Name`, t2.amount AS Amount, t2.time AS `Transaction Time`

ORDER BY `Transaction Time` DESC



1. **Common Merchants in Fraudulent and Legitimate Transactions**

MATCH (u:USER) -[:PERFORMED\_BY]-> (t1:TRANSACTION) -[:TO]-> (m1:MERCHANT)

WHERE t1.is\_disputed = '1'

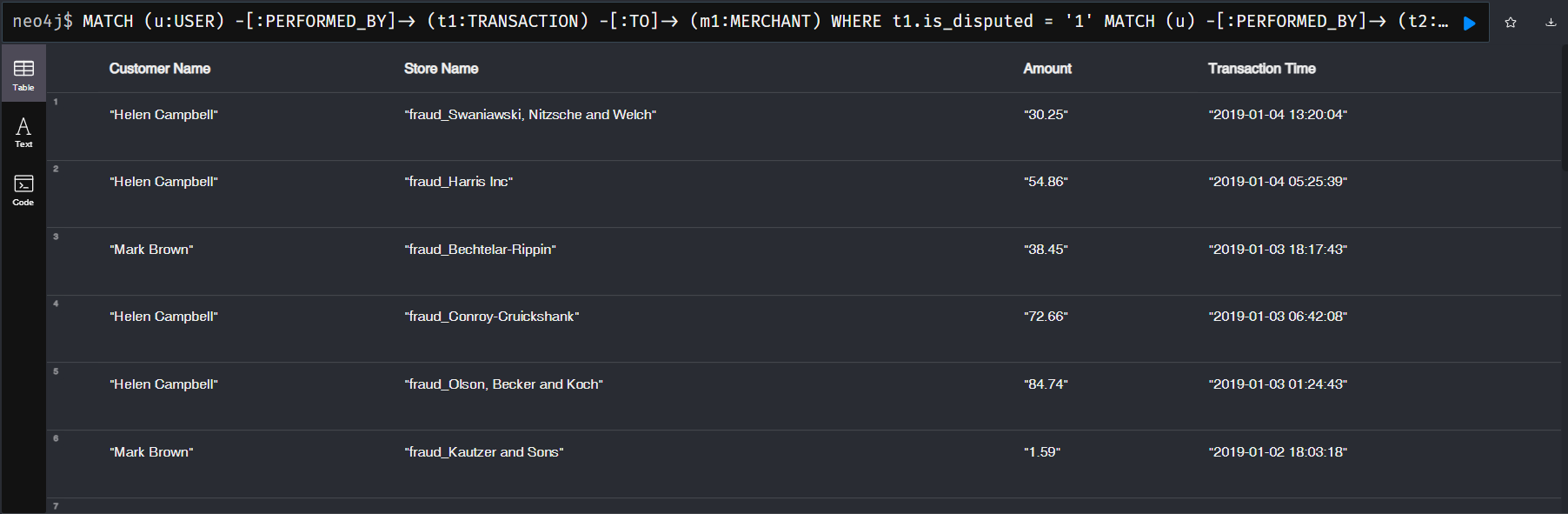
MATCH (u) -[:PERFORMED\_BY]-> (t2:TRANSACTION) -[:TO]-> (m2:MERCHANT)

WHERE t2.is\_disputed = '0' AND t2.time < t1.time

WITH u, m2, t2 ORDER BY t2.time DESC

RETURN DISTINCT m2.name AS `Suspicious Store`, count(DISTINCT t2) AS Count, collect(DISTINCT u.name) AS Victims

ORDER BY Count DESC



Conclusion

Using Neo4j, we were able to visualize the different relationships between users, transactions and merchants, find common merchants involved in different fraud cases, find relationship between victims and merchants involved in a fraud case and get data related to the fraud cases.

We were able to uncover difficult-to-detect patterns that far outstrips the power of relational database.