neo4j-contrib / **gists** Public

Code Issues Pull requests 1 Actions Projects Wiki Security Insights

master gists / other / BankFraudDetection.adoc Go to file ···

kbastani Ease of use Latest commit e3ac344 on Dec 5, 2013 **History**

1 contributor

Bank Fraud Detection

This interactive Neo4j graph tutorial covers bank fraud detection scenarios.

Table of Contents

Introduction to Problem

Banks and Insurance companies lose billions of dollars every year to fraud. Traditional methods of fraud detection play an important role in minimizing these losses. However increasingly sophisticated fraudsters have developed a variety of ways to elude discovery, both by working together, and by leveraging various other means of constructing false identities.

Explanation of Scenario

While no fraud prevention measures can ever be perfect, significant opportunity for improvement lies in looking beyond the individual data points, to the connections that link them. Oftentimes these connections go unnoticed until it is too late— something that is unfortunate, as these connections oftentimes hold the best clues.

Typical Scenario

While the exact details behind each first-party fraud collusion vary from operation to operation, the pattern below illustrates how fraud rings commonly operate:

- A group of two or more people organize into a fraud ring
- The ring shares a subset of legitimate contact information, for example phone numbers and addresses, combining them to create a number of synthetic identities
- Ring members open accounts using these synthetic identities
- New accounts are added to the original ones: unsecured credit lines, credit cards, overdraft protection, personal loans, etc.
- The accounts are used as normally, with regular purchases and timely payments
- Banks increase the revolving credit lines over time, due to the observed responsible credit behavior
- One day the ring “busts out”, coordinating their activity, maxing out all of their credit lines, and disappearing
- Sometimes fraudsters will go a step further and bring all of their balances to zero using fake checks immediately before the prior step, doubling the damage
- Collections processes ensue, but agents are never able to reach the fraudster
- The uncollectible debt is written off

Explanation of Solution

Graph databases offer new methods of uncovering fraud rings and other sophisticated scams with a high-level of accuracy, and are capable of stopping advanced fraud scenarios in real-time.

How Graph Databases Can Help

Augmenting one’s existing fraud detection infrastructure to support ring detection can be done by running appropriate entity link analysis queries using a graph database, and running checks during key stages in the customer & account lifecycle, such as:

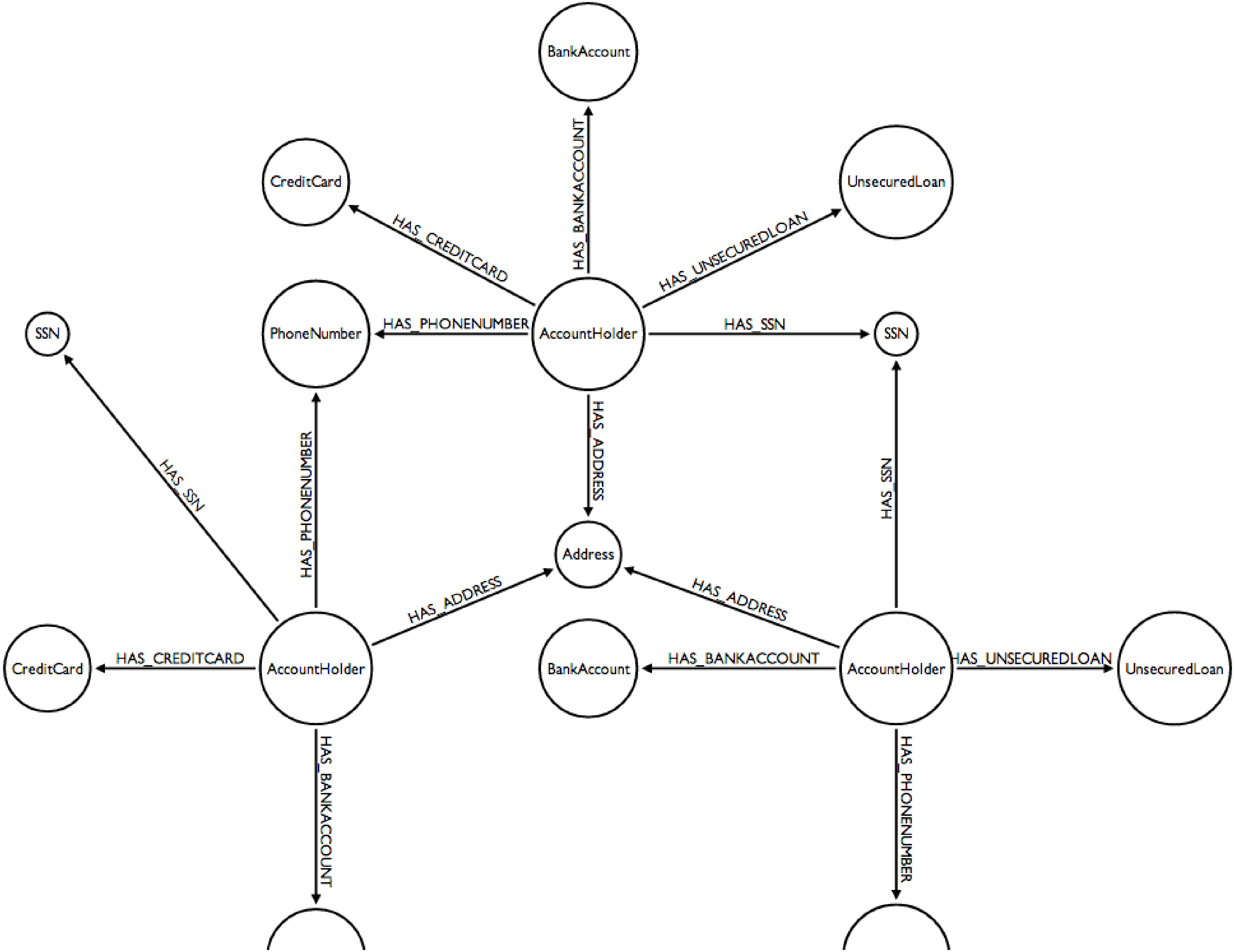
- At the time the account is created
- During an investigation
- As soon as a credit balance threshold is hit
- When a check is bounced

Real-time graph traversals tied to the right kinds of events can help banks identify probable fraud rings: during or even before the Bust-Out occurs.

Bank Fraud Graph Data Model

Graph databases have emerged as an ideal tool for overcoming these hurdles. Languages like Cypher provide a simple semantic for detecting rings in the graph, navigating connections in memory, in real time.

The graph data model below represents how the data actually looks to the graph database, and illustrates how one can find rings by simply walking the graph:


BankAccount
CreditCard
UnsecuredLoan
HAS_CREDITCARD
HAS_BANKACCOUNT
HAS_UNSECUREDLOAN
SSN
PhoneNumber
HAS_PHONENUMBER
AccountHolder
HAS_SSN
SSN
HAS_SSN
HAS_PHONENUMBER
HAS_ADDRESS
HAS_SSN
Address
HAS_ADDRESS
HAS_ADDRESS
CreditCard
HAS_CREDITCARD
AccountHolder
BankAccount
HAS_BANKACCOUNT
AccountHolder
HAS_UNSECUREDLOAN
UnsecuredLoan
HAS_BANKACCOUNT
HAS_PHONENUMBER

Sample Data Set

```
// Create account holders
CREATE (accountHolder1:AccountHolder {
                FirstName: "John",
                LastName: "Doe",
                UniqueId: "JohnDoe" })

CREATE (accountHolder2:AccountHolder {
                        FirstName: "Jane",
                        LastName: "Appleseed",
                        UniqueId: "JaneAppleseed" })

CREATE (accountHolder3:AccountHolder {
                        FirstName: "Matt",
                        LastName: "Smith",
                        UniqueId: "MattSmith" })

// Create Address
CREATE (address1:Address {
                        Street: "123 NW 1st Street",
                        City: "San Francisco",
                        State: "California",
                        ZipCode: "94101" })

// Connect 3 account holders to 1 address
CREATE (accountHolder1)-[:HAS_ADDRESS]->(address1),
       (accountHolder2)-[:HAS_ADDRESS]->(address1),
       (accountHolder3)-[:HAS_ADDRESS]->(address1)

// Create Phone Number
CREATE (phoneNumber1:PhoneNumber { PhoneNumber: "555-555-5555" })
```

```
// Connect 2 account holders to 1 phone number
CREATE (accountHolder1)-[:HAS_PHONENUMBER]->(phoneNumber1),
       (accountHolder2)-[:HAS_PHONENUMBER]->(phoneNumber1)

// Create SSN
CREATE (ssn1:SSN { SSN: "241-23-1234" })

// Connect 2 account holders to 1 SSN
CREATE (accountHolder2)-[:HAS_SSN]->(ssn1),
       (accountHolder3)-[:HAS_SSN]->(ssn1)

// Create SSN and connect 1 account holder
CREATE (ssn2:SSN { SSN: "241-23-4567" })<-[:HAS_SSN]-(accountHolder1)

// Create Credit Card and connect 1 account holder
CREATE (creditCard1:CreditCard {
```

239 lines (182 sloc) | 8.67 KB

Raw Blame

```
                        ExpirationDate: "01-20",
                        SecurityCode: "123" })<-[:HAS_CREDITCARD]-(accountHolder1)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount1:BankAccount {
                        AccountNumber: "2345678901234567",
                        Balance: 7054.43 })<-[:HAS_BANKACCOUNT]-(accountHolder1)

// Create Credit Card and connect 1 account holder
CREATE (creditCard2:CreditCard {
                        AccountNumber: "1234567890123456",
                        Limit: 4000, Balance: 2345.56,
                        ExpirationDate: "02-20",
                        SecurityCode: "456" })<-[:HAS_CREDITCARD]-(accountHolder2)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount2:BankAccount {
                        AccountNumber: "3456789012345678",
                        Balance: 4231.12 })<-[:HAS_BANKACCOUNT]-(accountHolder2)

// Create Unsecured Loan and connect 1 account holder
CREATE (unsecuredLoan2:UnsecuredLoan {
                        AccountNumber: "4567890123456789-0",
                        Balance: 9045.53,
                        APR: .0541,
                        LoanAmount: 12000.00 })<-[:HAS_UNSECUREDLOAN]-(accountHolder2)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount3:BankAccount {
                        AccountNumber: "4567890123456789",
                        Balance: 12345.45 })<-[:HAS_BANKACCOUNT]-(accountHolder3)

// Create Unsecured Loan and connect 1 account holder
CREATE (unsecuredLoan3:UnsecuredLoan {
                        AccountNumber: "5678901234567890-0",
                        Balance: 16341.95, APR: .0341,
                        LoanAmount: 22000.00 })<-[:HAS_UNSECUREDLOAN]-(accountHolder3)

// Create Phone Number and connect 1 account holder
CREATE (phoneNumber2:PhoneNumber {
                        PhoneNumber: "555-555-1234" })<-[:HAS_PHONENUMBER]-(accountHolder3)

RETURN *
```

Entity Link Analysis

Performing entity link analysis on the above data model is demonstrated below.

Find account holders who share more than one piece of legitimate contact information

```
MATCH           (accountHolder:AccountHolder)-[]->(contactInformation)
WITH            contactInformation,
                        count(accountHolder) AS RingSize
MATCH           (contactInformation)<-[]-(accountHolder)
WITH            collect(accountHolder.UniqueId) AS AccountHolders,
                        contactInformation, RingSize
WHERE           RingSize > 1
RETURN          AccountHolders AS FraudRing,
                        labels(contactInformation) AS ContactType,
                        RingSize
ORDER BY        RingSize DESC
```

Determine the financial risk of a possible fraud ring

```
MATCH           (accountHolder:AccountHolder)-[]->(contactInformation)
WITH            contactInformation,
                        count(accountHolder) AS RingSize
MATCH           (contactInformation)<-[]-(accountHolder),
                        (accountHolder)-[r:HAS_CREDITCARD|HAS_UNSECUREDLOAN]->(unsecuredAccount)
WITH            collect(DISTINCT accountHolder.UniqueId) AS AccountHolders,
                        contactInformation, RingSize,
                        SUM(CASE type(r)
                                WHEN 'HAS_CREDITCARD' THEN unsecuredAccount.Limit
                                WHEN 'HAS_UNSECUREDLOAN' THEN unsecuredAccount.Balance
                                ELSE 0
                        END) as FinancialRisk
WHERE           RingSize > 1
RETURN          AccountHolders AS FraudRing,
                        labels(contactInformation) AS ContactType,
                        RingSize,
                        round(FinancialRisk) as FinancialRisk
ORDER BY        FinancialRisk DESC
```