TCC - Bacharelado em Ciência da Computação

Detecção de fraudes pós-fato em operações cambiais baseada em análises de grafos

IME-USP

Victor Oreliana Fernandes Faria

Universidade de São Paulo

Brasil - 2019

Meus sinceros agradecimentos às suas atenciosas orientações,

Profa. Dra. Kelly Rosa Braghetto DCC IME USP - orientadora

Márcio Willian Melo Caldeira, CTO - BeeTech Global - co-orientador

Profa. Dra. Nina Sumiko Tomita Hirata - DCC IME USP - coordenação geral

Índice

1 - **Introdução**

1.1 **Contexto e Motivação**

Segundo a atual normativa do Banco Central do Brasil (BACEN) a respeito do mercado de câmbio [1], cabe às instituições atuantes zelar pela conformidade de remessas de capital ao exterior. Esta regulação traz para as instituições cambiais a responsabilidade de desenvolver métodos analíticos que processem informações de remessas relativas ao agente remetente, ao agente recebedor e à transação em si. Assim, essas instituições podem atuar como órgãos autorreguladores que fiscalizam a transação sobre os aspectos tributário e criminal, enviando relatórios aos devidos órgãos públicos responsáveis respectivos, a Secretaria da Receita Federal do Brasil e BACEN. Tal fiscalização é fundamental para:

i) dificultar e prevenir o financiamento de organizações criminosas. OLIVEIRA [2] mostra a relevância do câmbio no âmbito criminal; e,

ii) para reduzir a sonegação tributária - são arrecadados bilhões de reais em tais operações anualmente [3] e, portanto, mesmo um pequeno aumento percentual é bastante significativo.

A análise de conformidade é tipificada em duas categorias, pré-fato e pós-fato. A primeira categoria contempla a identificação de suspeitas de fraude de forma tal que as informações de uma única remessa são necessárias e suficientes para apontá-la como suspeita. A segunda contempla a identificação de suspeitas de fraude através da análise da relação entre conjuntos de atributos de duas ou mais remessas. Análises pré-fato são computacionalmente simples pois podem ser realizadas confrontando um conjunto de regras estruturadas sobre uma modelagem relacional de dados ao dado de entrada - e.g. uma pessoa física realizando uma remessa para uma pessoa jurídica declarando estar fazendo-a para outra pessoa física a fim de se eximir de tributações comerciais. No entanto, as análises pós-fato são computacionalmente interessantes.

1.2 Objetivos Gerais

Os objetivos gerais do trabalho consistem em aplicar algoritmos em grafos para identificação de padrões de suspeitas de fraudes em operações cambiais pós-fato. Elencamos a seguir quatro casos de interessante abordagem.

1. O caso de lavagem de dinheiro através de co-autores. Dada a existência de um montante ilícito de recursos a serem remetidos ao exterior, este pode ser remetido por um número, N, de co-autores sem precedentes criminais. Neste caso, o valor total a ser remetido é dividido entre os co-autores de forma que cada um realize uma remessa de baixo valor, aparentemente em conformidade (remessas de baixo valor possuem menos critérios de fiscalização e, para o BACEN, um baixo valor é qualquer valor até o equivalente em reais a 3 mil dólares americanos). Sendo T o número de linhas de uma tabela de transações financeiras em um banco de dados relacional, cada busca por suspeitas de fraudes deste tipo irá requerer a varredura de toda a tabela resultando em uma complexidade de tempo O(T) e, portanto, a emissão de um relatório contemplando todos os casos possíveis resultará em uma complexidade de tempo $O(T^2)$, sem considerar ainda o custo da execução de regras para retirar repetições.

2. O caso de sonegação fiscal através de um anel de fraude. Um anel de fraude é definido por agentes operando em conjunto, individualmente dentro da normalidade, porém cujas interações resultam em fraudes. Por exemplo, considere um caso em que 3 agentes (A, B e C) realizam remessas de igual valor de forma que A envia para B, B para C e C para A. Esse caso é equivalente a cada um dos agentes enviar uma remessa para si, entretanto o regime tributário do primeiro caso resulta ganho de 289,47% em impostos não arrecadados quando comparado ao segundo, pois a alíquota de IOF (Imposto sobre Operações Financeiras) no primeiro caso é de 1,1% e no segundo de 0,38%. Convencionalmente, em um banco de dados relacional, uma consulta para revelar tais anéis de fraude consumiria tempo O(N^T), onde N é o tamanho do anel e T é o número de linhas da tabela de transações financeiras, por ser necessário realizar

um produto cartesiano (TxT) a cada iteração i (1 <= i <= N) que compõem a identificação do anel. Além disto, como no caso anterior, haveria também o custo da execução de regras para remover repetições.

3. O caso do dreno de capacidade financeira. Consideramos que um agente recebedor é um dreno de capacidade financeira caso ele consuma todo o limite operacional de um agente remetente. Podemos então aplicar uma correlação histórica entre os agentes remetentes e o agente recebedor. Ou seja, mesmo que seja aferida a conformidade de remessas entre cada par (remetente-recebedor), é considerado suspeito que ao longo de um ano fiscal um agente recebedor receba toda a capacidade financeira de múltiplos agentes remetentes. Este caso difere do (1) pois não é necessário um número relativamente grande de remetentes para diluição da remessa - um ou dois agentes remetentes configurando este padrão é suficiente para motivar uma investigação mais detalhada, ainda com o agravante do valor total transacionado. A dificuldade de análise deste caso é que ele é composto de muitas remessas entre poucos agentes que são espalhadas ao longo do ano.

4. O caso do intermediário de alta capacidade financeira. Considere dois conjuntos de nós com os quais deseja-se formar um grafo bipartido, no entanto, não é possível que haja uma conexão direta entre nós desses dois conjuntos. Neste caso é adicionado um nó intermediário entre os dois conjuntos, o qual serve de ponte de conexão entre ambos. Este pode ser o caso de uma importação ou exportação de recursos não licenciados, por exemplo, uma companhia tem apenas licença para fazer importações de um tipo hipotético de mercadorias, mas deseja importar um segundo tipo - neste caso cada conjunto de nós referido acima trata-se ou do conjunto de importadores ou de exportadores, e o nó intermediário é um agente fraudador que possui ambas as licenças e serve de atravessador trocando a natureza das importações. Neste caso podemos observar que o nó intermediário terá uma medida de centralidade [17] anômala

A fim de lidar efetivamente com esses problemas computacionais, propomos uma abordagem de processamento embasada em bancos de dados orientados a grafos cujo desempenho teórico do tempo de consulta é otimizado e cuja expressão de linguagem de consulta é mais simples. De fato, encontramos casos de sucesso na literatura que são similares a respeito de modelagens de fraudes com grafos [4][5].

1.3 **Objetivos Específicos**

Estudaremos neste texto o caso de fraudes pós-fato de operações de câmbio no Brasil cujos remetentes são pessoas físicas, nos restringindo à identificação e modelagem de padrões de grafos e de um sistema de pontuação sobre os agentes. Os grafos deverão envolver pessoas e transações e o sistema de pontuação diz respeito ao cálculo de uma média ponderada de uma pontuação atribuída a padrões de grafos encontrados e a indicadores de estatística descritiva dos agentes; esta média será usada para classificar o risco potencial de uma transação de acordo com uma escala configurável. Para tanto, listamos a seguinte lista de objetivos específicos para este fim.

1. Desenvolver uma modelagem de grafos na qual seja possível a identificação de padrões que contemplem fraudes de câmbio.

2. Propor um sistema de monitoramento de transações computacionalmente eficiente que seja capaz de identificar padrões de fraudes cambiais pós-fato e realizar classificações de risco.

3. Criar uma coleção de dados fictícia que permita aferir que o sistema realmente é capaz de detectar as fraudes propostas.

1.4. **Organização do texto**

O restante deste texto está organizado da seguinte forma:

1. Na seção 2 apresentamos os conceitos usados ao longo do desenvolvimento do texto.
2. Na seção 3 formalizamos as definições dos esquemas e modelagens de dados.
3. Na seção 4 formalizamos os padrões de fraudes que trabalharemos sobre e trabalhamos nas suas definições de identificação.
4. Na seção 5 definimos uma ferramenta de software para detectar fraudes sistematicamente em um ambiente de produção.
5. Na seção 6 apresentamos um estudo de caso referente a consultas sobre a modelagem proposta.
6. Na seção 7 declaramos as conclusões finais do texto.

2. **Conceitos**

Buscamos aqui recapitular ou introduzir conceitos relacionados a bancos de dados relacionais [20], a bancos de dados orientados a grafos [21], às linguagens de consulta SQL e Cypher e a grafos de propriedades.

Bancos de dados relacionais têm sido por décadas a solução padrão para armazenamento de dados em sistemas de software. Isto muito devido à sua robustez em garantir a integridade de dados, as propriedades ACID (*atomicity, consistency, isolation, and durability*) [23], o uso eficiente de memória, o rápido acesso à buscas através das estruturas de índice e à expressividade da SQL - que é uma linguagem específica de domínio usada na programação e projetada para gerenciar dados mantidos em um sistema de gerenciamento de banco de dados relacional (SGBDR). É particularmente útil no tratamento de dados

estruturados, isto é, dados que incorporam relações entre entidades e variáveis [24]. No entanto, mais recentemente, não apenas as aplicações têm se tornado cada vez mais conectadas mas também tem-se requerido cada vez mais acesso aos relacionamentos dos dados dentro de uma mesma aplicação.

É em meio à essa demanda de hiperconectividade que tem se popularizado soluções alternativas para armazenamento de dados. Consultas SQL envolvendo múltiplas junções *(JOINs)* e/ou agregações em tabelas com volumetria na ordem de milhões de linhas possuem algumas características inconvenientes, tais como: necessidade de varredura em múltiplas tabelas, alta capacidade de memória e processamento requeridas em tempo de execução. Esses recursos computacionais são muito relevantes no cenário de aplicações em nuvem, no qual o custo de manter o sistema é derivado da capacidade computacional empregada e do número de máquinas necessário para garantir a disponibilidade do sistema em função do consumo da aplicação ao longo do tempo. Manter uma disponibilidade saudável exige ainda um tanto a mais de recursos, ou seja, mais custos.

Neste cenário é importante entendermos quais as características do sistema de software em questão e fazer uso otimizado das ferramentas de armazenamento e consulta de dados - o que tem por sua vez aumentado a presença de aplicações com bancos de dados poliglotas, ou seja, aplicações que usam mais de um tipo de banco de dados para persistirem seus dados. Neste caso temos nos preocupado com um sistema de alta volumetria entretanto com poucos relacionamentos entre suas tuplas, isto do ponto de vista de uma modelagem relacional, e através de uma modelagem orientada a grafos, este sistema tem dados cuja principal característica é o seu baixo grau de centralidade [17].

Desta demanda por melhor relação entre armazenamento e consumo de dados é que partimos para analisar, em particular, os bancos de dados orientados a grafos - embora outras abordagens tais como “orientado a documento”, “chave-valor” e “orientado a colunas”

também estejam disponíveis. Temos como respectivos exemplos os sistemas gerenciais das abordagens citadas: MongoDB, Redis e Cassandra.

Vamos trabalhar com o popular Sistema Gerenciador de Bancos de Dados Orientado a Grafos (SGBDOG) que, como tal, lida com os relacionamentos entre os dados mais eficientemente que um Sistema Gerenciador de Bancos de Dados Relacional (SGBDR), pois o relacionamento é intrínseco à modelagem dos dados e exige pouco processamento para extrair informações de relacionamentos. Por exemplo, em um SGBDR em uma consulta sobre uma relação muitos-para-muitos é necessário realizar junções em, ao menos, três tabelas, ou seja, três varreduras. Nesse SGBDOG, por outro lado, seria realizada apenas uma varredura para encontrar os nós, o acesso a todos os outros nós relacionados seria feito a partir de um lista ligada associada ao primeiro conjunto de nós encontrado. O artigo [18], mostra bem a conversão de um modelo relacional para um modelo orientado a grafo. Abaixo trazemos um excerto traduzido deste artigo para aprofundarmos a contextualização.

> SBGDRs armazenam dados altamente estruturados em tabelas com colunas predeterminadas de tipos específicos e muitas linhas desses tipos de informações definidos. Devido à rigidez de sua organização, os bancos de dados relacionais exigem que os desenvolvedores e aplicativos estruturem estritamente os dados usados em seus aplicativos.
>
> Nos bancos de dados relacionais, as referências a outras linhas e tabelas são indicadas consultando os atributos da chave primária por meio de colunas de chave estrangeira. As junções são calculadas no momento da consulta, combinando chaves primárias e estrangeiras de todas as linhas nas tabelas conectadas. Essas operações exigem muita computação e muita memória e têm um custo exponencial.
>
> Quando relacionamentos muitos-para-muitos ocorrem no modelo, você deve apresentar uma tabela JOIN (ou tabela de entidade associativa) que contém chaves estrangeiras de ambas as tabelas participantes, aumentando ainda mais os custos de operação da união. A imagem abaixo mostra esse conceito de conexão de uma Pessoa (da tabela Pessoa) a um Departamento (na tabela Departamento), criando uma tabela

de junção Pessoa-Departamento que contém o ID da pessoa em uma coluna e o ID do departamento associado na próxima coluna.

Como você provavelmente pode ver, isso torna o entendimento das conexões muito complicado, pois é necessário conhecer os valores de ID da pessoa e do departamento (executando pesquisas adicionais para encontrá-los) para saber qual pessoa se conecta a quais departamentos. Esses tipos de operações caras de junção geralmente são resolvidos desnormalizando os dados para reduzir o número de junções necessárias, quebrando a integridade dos dados de um banco de dados relacional.

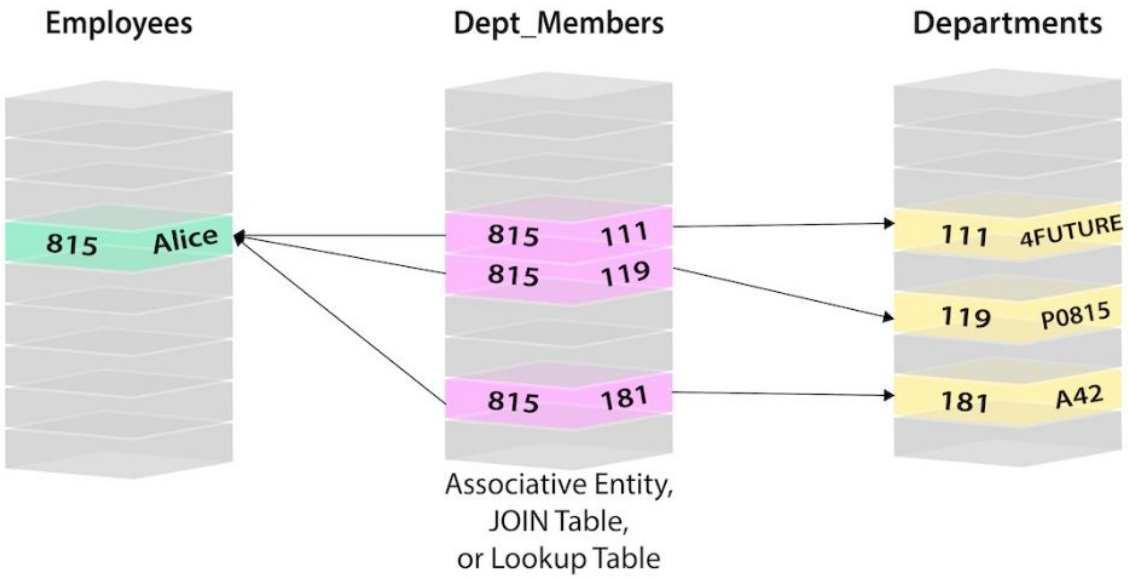

Fig A. Relacionamento muitos para muitos em um banco de dados relacional, retirado de https://neo4j.com/developer/graph-db-vs-rdbms/, em Agosto de 2019

Embora nem todos os casos de uso sejam adequados para esse modelo de dados rigoroso, a falta de alternativas viáveis e o amplo suporte para bancos de dados relacionais dificultavam a entrada de modelos alternativos no mainstream. No entanto, a era NoSQL chegou ao mercado, preenchendo algumas necessidades de usuários e empresas, mas ainda perdendo a importância das conexões entre os dados. Foi assim

que nasceram os bancos de dados orientados a grafos. Eles foram projetados para oferecer a maior vantagem no mundo conectado em que vivemos hoje.

No exemplo relacional acima, pesquisamos a tabela Pessoa à esquerda (potencialmente milhões de linhas) para encontrar o usuário Alice e seu ID de pessoa 815. Em seguida, pesquisamos a tabela Pessoa-Departamento (tabela do meio laranja) para localizar todas as linhas que fazem referência à identificação da pessoa de Alice (815). Depois de recuperar as três linhas relevantes, vamos para a tabela Departamento, à direita, para pesquisar os valores reais dos IDs de departamento (111, 119, 181). Agora sabemos que Alice faz parte dos departamentos 4Future, P0815 e A42.

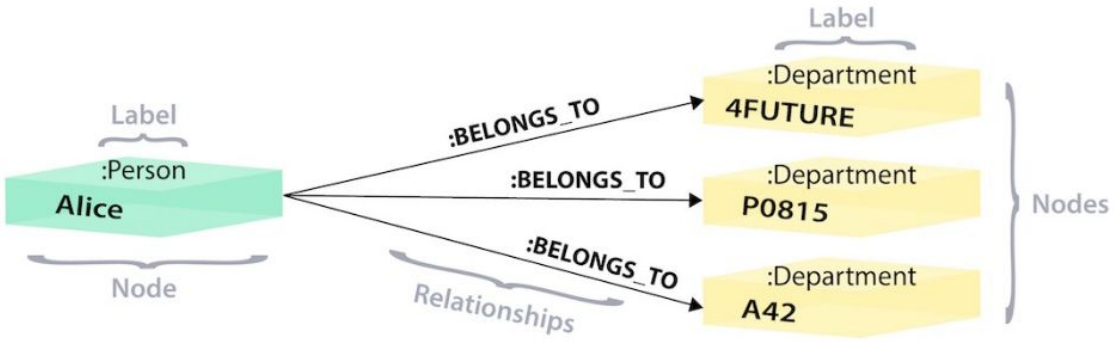

Fig B. Relacionamento entre nós de um grafo em um banco de dados orientado a grafos, retirado de https://neo4j.com/developer/graph-db-vs-rdbms/, em Agosto de 2019

Na versão em grafos acima, temos um único nó para Alice com um rótulo de Pessoa. Como Alice pertence a 3 departamentos diferentes, criamos um nó para cada um e com um rótulo de Departamento. Para descobrir a quais departamentos Alice pertence, procuraríamos no grafo o nó de Alice e percorreríamos todos os relacionamentos BELONGS_TO de Alice para encontrar os nós de departamento aos quais ela está conectada. É tudo o que precisamos: um único salto sem pesquisas envolvidas.

O SGBDOG que adotamos neste texto é o Neo4J. Escolhemos pontualmente este devido ao fato dele implementar a linguagem de consulta Cypher e usar o modelo de *grafos de propriedades* [19]. Esta linguagem propõe um padrão para a consulta a bancos de dados orientados a grafos, analogamente à SQL em relação aos SGBDRs. Entendemos que a Cypher está bem madura [22] e é a melhor alternativa atualmente para lidar com implementações de soluções em bancos de dados orientados a grafos.

> *Cypher é uma linguagem declarativa de consulta de grafos que permite consultas e atualizações expressivas e eficientes do grafo. Ela foi projetada para ser adequado para desenvolvedores e profissionais de operações. O Cypher foi projetado para ser simples, mas poderoso; consultas de banco de dados altamente complicadas podem ser facilmente expressas, permitindo que você se concentre no seu domínio, em vez de se perder no acesso ao banco de dados. - The Neo4j Cypher Manual v3.5 [22]*

Grafos de propriedades são objetos tais como grafos convencionais, definidos através de nós e arestas, com o diferencial de um terceiro elemento em meio a estes dois. Este elemento é a lista de propriedades, que é uma tabela de símbolos (chave-valor) e pode ser relacionada a um nó ou a uma aresta. Desta forma é possível, e em nosso caso conveniente, estabelecermos consultas tendo cláusulas de condição que referenciam a lista de propriedades de nós e arestas

O modelo físico de armazenamento de dados do Neo4J é bastante simples. São armazenados em disco separadamente: nós, relacionamentos e propriedades - sendo cada nó, relacionamento ou propriedade um arquivo. Cada nó é associado a um conjunto de relacionamentos, propriedades e etiquetas (labels). Os relacionamentos são compostos por listas ligadas que conectam os nós e possuem propriedades. Cada propriedade possui um tipo e cada tipo aloca uma quantidade fixa de memória - essa memória é separada em blocos de

tamanhos específicos e cada tipo pode possuir um ou mais blocos para compor a quantidade de memória por ele requerida.

Por conta deste modelo físico baseada em listas ligadas que promove mais velocidade nas consultas, pois, dado que tenhamos um registro e desejamos encontrar um outro com ele relacionado, temos uma complexidade de tempo de busca O(N) em um cenário de pior caso - ou se consideramos a dada pelo sistema, descrito acima, de que temos alta volumetria e pouco grau de centralidade, a complexidade de tempo de busca no caso médio é O(1). Todavia em um SBGBDR teríamos sempre que realizar um produto cartesiano NxN para cada iteração de relacionamento, e como no pior caso podemos ter uma quantidade linear de iterações, temos que o tempo de busca no pior caso é O(N^I) - sendo I o número de iterações, que é análogo ao comprimento do grafo.

3. Banco de Dados para Transações de Câmbio

Apresentamos aqui a maneira com a qual trabalharemos os dados para atingirmos os objetivos apresentados no Capítulo 1. Buscamos o entendimento da natureza dos dados e a elaboração de esquemas de dados consoantes. Procura-se deixar evidente através desta seção a razão pela qual uma modelagem orientada a grafos é uma boa abordagem para o problema referido neste trabalho.

3.1 Descrição dos dados

Temos três entidades fortes: “Cliente”, “Beneficiário” e “Transação”. Vamos estabelecer a relação entre elas e com aspectos do problema de detecção de suspeitas de fraudes a fim de motivarmos as modelagens que seguem.

“Cliente” , em nosso escopo, é uma pessoa física que realiza uma remessa de envio. “Beneficiário” pode ser uma pessoa física ou um pessoa jurídica e é quem recebe a remessa no exterior. É importante sua relação com seus dados bancários pois seus dados bancários fazem parte de sua identidade, pois no processo de remessas não é possível conhecer dados pessoais do “Beneficiário” a menos de uma sobreposição com a entidade “Cliente”. Uma sobreposição é definida pela caracterização de um pessoa tanto quanto “Cliente” como “Beneficiário”. “Transação” diz respeito à própria remessa e é o que associa clientes a beneficiários - é possível dizer que Transação é uma relação de muitos para muitos entre clientes e beneficiários.

“Cliente” possui os dados cadastrais e operacionais de uma pessoa, isto é, seus dados pessoais e parâmetros operacionais (como, por exemplo, capacidade financeira, descontos etc.). “Beneficiário” detém informações bancárias e um substrato de dados pessoais usados para sustentar sua identidade. Por último, “Transação” possui referências para seus respectivos cliente e beneficiário bem como informações financeiras e cambiais.

3.2 Modelagem dos dados

Apresentamos aqui os dois esquemas de bancos de dados que temos interesse segundo cada uma das abordagens endereçadas, a relacional e a orientada a grafos. Ainda na seção 5.3 apresentaremos uma modelagem orientada a documentos motivada pela conveniência em relação à ferramenta de detecção de suspeitas de fraudes proposta - no entanto esta modelagem é meramente técnica, não tendo uma implicação teórica abordada neste capítulo.

3.2.1 Modelagem Relacional

Esta modelagem é conveniente para este problema pela seguinte perspectiva: é interessante para o problema que estamos tratando, termos acesso fácil a indicadores de

estatística descritiva derivados de consultas envolvendo funções de agregação - por exemplo, pode ser interessante saber quantas transações com determinada especificação foram realizadas em um determinado período. Além disto, a modelagem relacional é conveniente para estabelecermos uma relação entre as modelagens, isto é, podemos armazenar quais são as consultas Cypher envolvidas em uma ferramenta de detecção em tabelas. Estas consultas possuem a finalidade de dar suporte a uma análise do grafo através da ferramenta de detecção de suspeitas de fraudes.

Ou seja, teríamos o SGBDR estruturando recursos para embasar análises em geral. Um exemplo disto: seria possível mantermos diversos registros descrevendo as verificações mais adequadas para cada cenário de análise. O gerenciamento desses dados, de perfis de consultas associados a perfis de fraudes poderia ser gerido pela ferramenta proposta para alteramos o comportamento de detecção de fraudes em tempo de execução conforme fosse mais conveniente para conjuntos específicos de análises.

Também poderíamoss usar um SGBDR para persistir análises através de indicadores (nomeados aqui “flags”) permitindo assim extrair conhecimento da modelagem orientada a grafo sem necessariamente fazer uso de processamento computacional sobre a modelagem de grafos em tempo de consulta.

O esquema do banco de dados relacional é mostrado na Figura 1. Nesse esquema, definimos uma transação entre um cliente (tabela “*customer*”) e um beneficiário (tabela “*beneficiary*”) - sendo que o beneficiário pode, eventualmente, ser o próprio cliente. Além disto, ainda temos a tabela “*flag”*, cuja função é prover marcações relevantes sobre as entidades transação, cliente e beneficiário. Por exemplo, algumas dessas marcações poderiam ser: “suspeita de fraude”, “validado por analista” , “lucro acima da média”, etc.

A tabela “*graph query*” constitui um vínculo entre o banco de dados relacional e o banco de dados orientado a grafos. Nesta, listamos quais consultas estarão disponíveis para o sistema interagir com o banco de dados orientado a grafos. Como produto desta interação

teremos entradas na tabela "*report*", a qual sustenta a pontuação do grau de risco de uma operação. Em "*score level*" definimos uma escala de pontuação de risco através de intervalos.

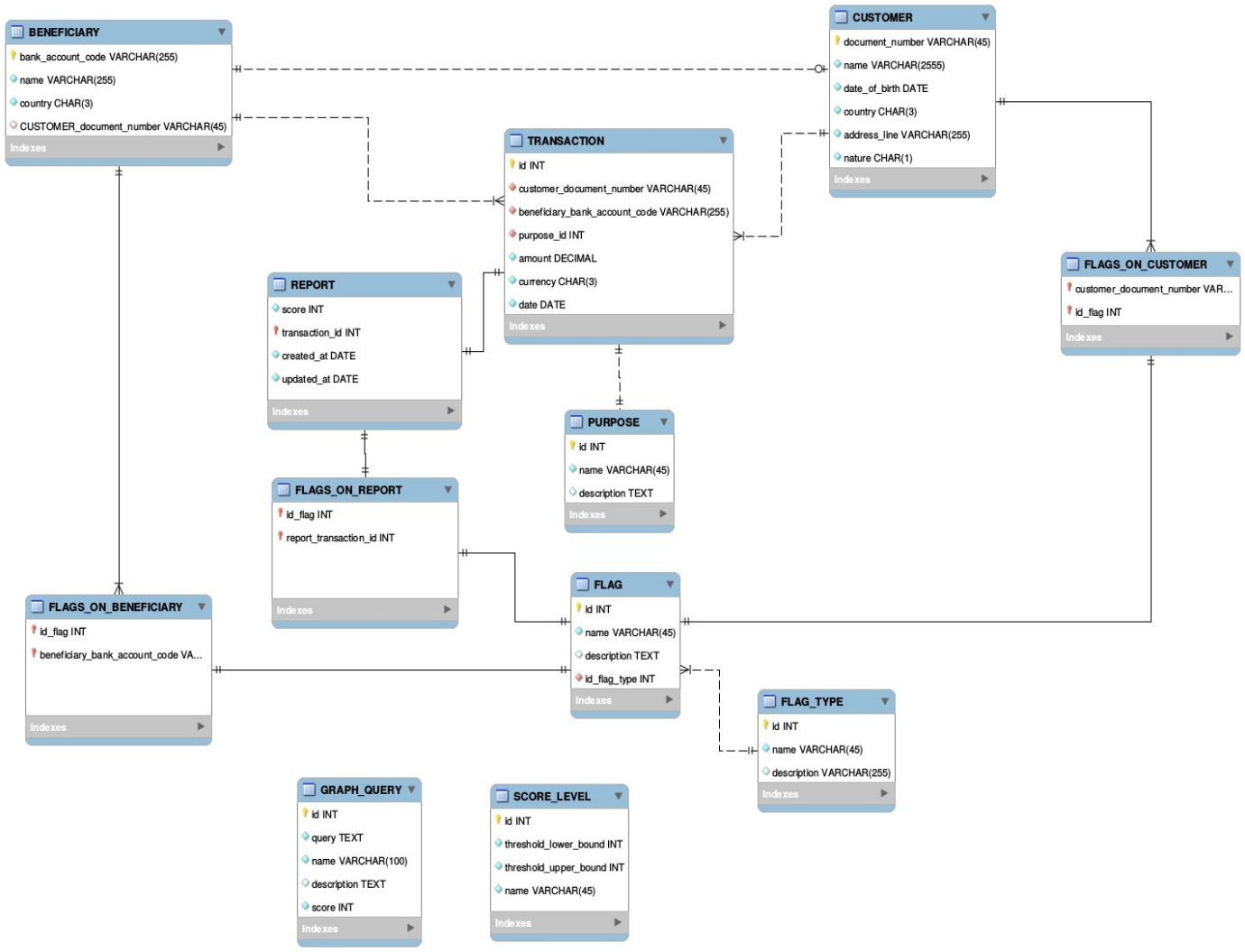

Fig. 1: Diagrama do esquema do banco de dados relacional

3.2.2 Modelagem Orientada a Grafos

Seguiremos aqui com a definição de um grafo de propriedades conveniente para nosso problema em questão. Definimos duas propriedades de nós que os categorizam de forma não excludente: *cliente* e *beneficiário*; e uma outra propriedade para arestas: *transação*. O sentido da aresta é sempre de *cliente* para *beneficiário* (o sentido da aresta pode ser expressado como uma propriedade da aresta) - isto é, o sentido da aresta é de um nó com propriedade *cliente* para um nó com propriedade *beneficiário*. É importante observar que muitos *beneficiários*

podem vir a se referir à mesma pessoa física de um *cliente*, enquanto *beneficiário*, por sua, vez, é a associação a uma conta bancária internacional em nome de uma pessoa física e poderia ser chamada de "conta de recebimentos do *beneficiário*", no entanto por simplicidade reduzimos o nome.

O BD orientado a grafo usado neste trabalho é um grafo de propriedades que segue o esquema (V, A, Pv, Pa) um grafo de propriedades, sendo que V é seu conjunto de vértices, A seu conjunto de arestas, Pv o conjunto de conjuntos de propriedades dos vértices e Pa o conjunto de conjuntos de propriedades das arestas. Pv = { 'cliente', 'beneficiário', atributosDoCliente, atributosDoBeneficiario}, Pa = { atributosDaTransação }, sendo que atributosDoBeneficiário é o subconjunto { 'nome', 'numeroDeConta', nomeDoBanco' } e atributosDoCliente é o subconjunto { 'numeroDoDocumento', 'nome,', 'tipo, 'pais', 'endereco}, atributosDaTransação = { 'id', 'quantidade, 'moeda, 'data', 'natureza}. Temos também as seguintes relações sobre as propriedades:

1. 'cliente' || 'beneficiário' = true
2. 'cliente' => atributosDoCliente != {}
3. 'beneficiário' => atributosDoBeneficiário != {}

As quais devemos ler da seguinte forma: 'propriedade', a referida propriedade existe, ou está presente no conjunto. Analogamente para subconjuntos. A seção 5.3 traz tais especificações sobre as propriedades na notação de JSONSchema.

4. Detecção de Suspeitas de Fraudes

Vamos aqui formalizar três padrões de fraudes segundo o grafo definido na seção 3.2.2, exemplificar consultas Cypher para extraí-los e definir uma heurística de análise de resultados visando facilitar a administração do dados encontrados através de priorizações. Reduzimos, então, o escopo de padrões possíveis (inclusive alguns elencados na seção 1.2) com a finalidade de ilustrar casos fundamentais, entretanto elaborar padrões diferentes fica a cargo das necessidades organizacionais.

4.1 Modelagem das Consultas para Identificar Padrões em Grafos

A primeira consulta visa identificar suspeitas de evasão de divisas. Pelas normas da regulamentação brasileira de câmbio, a natureza (ou propósito) da remessa implica em diferentes alíquotas tributárias. Uma pessoa realizando uma remessa para si mesmo tem uma alíquota maior do que esta mesma pessoa enviando remessa para um terceiro com vínculo familiar. Como o rastreamento de vínculo familiar não é trivial, muitos se aproveitam desta brecha para recolher menos impostos. Outro caso frequente acontece entre cônjuges, quando um membro do casal deseja fazer uma remessa para si mesmo mas acaba usando seu cônjuge para emular uma transferência com a finalidade "terceiro com vínculo familiar" ou "manutenção de residente". Segundo as normativas atuais, uma remessa para o mesmo portador possui uma taxação maior devido ao entendimento de haver recursos deixando o país do que uma remessa para um terceiro com vínculo familiar. A ocorrência desse padrão é exemplificada na fig. 2.

Definição 1: Grafo com padrão circular (de suspeita de evasão de divisas)

Seja G um grafo de propriedades no esquema definido na seção 3.2.2. G *contém um padrão circular* se ele contém um ciclo envolvendo pelo menos 2 nós ou, mais formalmente, se ele contém N nós {n_1, n_2, ... , n_N}, com N > 1, e N arestas {a_1, a_2, ... , a_N} tais que a_i = (n_i , n_i + 1 mod N) para 1 <= i <= N. A ocorrência do padrão circular aponta uma suspeita de evasão de divisas.

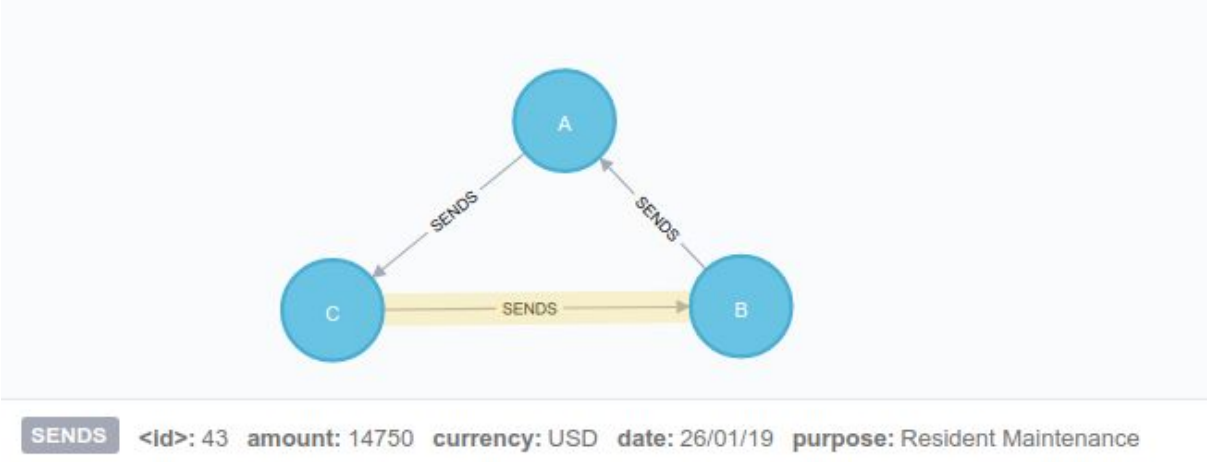

Fig 2. Grafo exemplificando um padrão circular com suspeita de evasão de divisas com três agentes. A natureza das transações são “terceiro com vínculo familiar”. O atributo *amount* na aresta significa a quantidade em moeda estrangeira sendo transferida.

A segunda consulta visa identificar suspeitas de lavagem de dinheiro ou evasão de divisas. A interpretação para lavagem de dinheiro é a de que um agente possui uma grande soma de valores ilícitos e a distribui para vários terceiros a fim de eles fazerem remessas para uma conta sua no exterior. A interpretação para evasão de divisas é que há uma finalidade comercial, ou seja, um agente está recebendo pagamentos através de remessas com naturezas diferentes de naturezas comerciais e assim recolhendo menos tributos. A ocorrência desse padrão é exemplificada na fig. 3.

Definição 2: Grafo com padrão “1-N” (de suspeita de lavagem de dinheiro ou evasão de divisas)

Seja G um grafo de propriedades no esquema definido na seção 3.2.2. G *contém padrão “1-N”* se ele contém N nós {n_1, n_2, ... , n_N}, com N > 1, e N-1 arestas {a_1, a_2, ... , a_N-1}, tais que a_i = (n_i , n_N) para 1 <= i < N. A ocorrência do padrão “1-N” aponta suspeita de lavagem de dinheiro ou evasão de divisas.

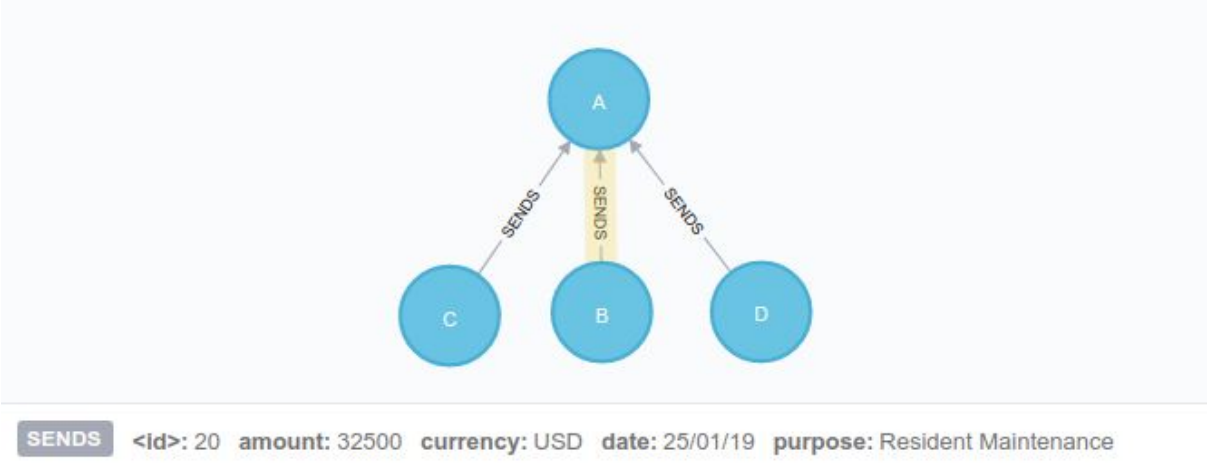

fig 3. Grafo exemplificando um padrão (1-N) com suspeita de lavagem de dinheiro ou evasão de divisas com quatro agentes. O número na aresta significa a quantidade em moeda estrangeira sendo transferida.

4.2 Consultas para Detecção de Fraudes

Definimos abaixo consultas de dados especificadas na linguagem Cypher que correspondem aos padrões de fraudes definidos acima.

```
1. $ MATCH(customer)-[*1..2]->(intermediary)-[*1..2]->(customer)
2. RETURN *
```

Consulta (1) para fraude do padrão circular

Esta consulta busca por um padrão circular envolvendo de um a três intermediários, ou seja, de dois a quatro nós. É importante delimitar a quantidade de intermediários quando o grafo armazenado tem muitas conexões em relação ao número de nós, pois pode haver uma explosão combinatória que impactaria no tempo consumido pela consulta. Devemos ler essa consulta como "encontre a partir de nós quaisquer com a propriedade 'customer' passando por um ou dois outros nós quaisquer nós a partir deste, chegando a um intermediário diferente do primeiro nó e passando novamente por até outros um ou dois nós quaisquer e chegando finalmente ao primeiro nó e retornando todos os nós envolvidos como resultado".

```
1. $ MATCH(customer:Customer)-[r:SENDS {purpose:
2. 'Resident Maintenance'}]-(beneficiary:Beneficiary) WITH
3. customer, COUNT(beneficiary) AS beneficiaries WHERE
4. beneficiaries >= 3 RETURN customer
```

Consulta (2) para fraude do padrão um para muitos

```
$ MATCH(customer:Customer)-[r:SENDS {purpose:
'Resident Maintenance'}]-(beneficiary:Beneficiary) WITH
beneficiary, COUNT(customer) AS customers WHERE
customers >= 3 RETURN beneficiary
```

Consulta (3) básica para fraude de padrão muitos para um

Note que nestas duas consultas acima definimos N=3 como limiar indicativo de uma quantidade suficientemente grande para classificar uma suspeita de fraude em ambas consultas acima, no entanto tal parâmetro deve ser regulado por especialistas de acordo com análises de dados históricos e uma limitação viável do tempo de processamento esperado. Dependendo do grau de conectividade do grafo e do número de elementos, pode haver dificuldades de escalar N para um limiar tão alto quanto o desejado em um intervalo de tempo aceitável para determinadas análises que dependem de velocidade.

4.3 Heurística de pontuação

O propósito desta seção é definir um modelo de classificação de risco a fim de auxiliar a análise especialista de um conjunto de potenciais fraudes detectadas. Desta maneira espera-se que a chance de detecção de fraude na análise pós-fato seja maior dada a limitação imposta pelo custo em recursos humanos de uma análise.

Definimos tal modelo como sendo uma escala ordenada, na qual 1 é menos relevante que 2, que por sua vez é menos relevante do que 3 e assim por diante, definida heuristicamente e baseada tanto em parâmetros definidos pelo BACEN quanto na observação empírica cotidiana dos especialistas. Escala:

1. Qualquer combinação de padrão de fraude
2. Qualquer combinação de padrão de fraude com valor transacionado maior ou igual U$ 3.000,00 ou equivalente em outras moedas, por operação

3. Padrões 1-N e N-1 com operações cuja soma agregada seja maior ou igual a U$ 3.000,00 ou equivalente em outras moedas
4. Padrões 1-N e N-1 com operações cuja soma agregada seja maior ou igual a U$ 3.000,00 ou equivalente em outras moedas, com N >= 3

Para cada um dos itens acima, de 1 a 4, podemos adicionar a condição "entre operações realizadas em datas de criação limitadas a 90 dias de distância entre si", assim definindo, respectivamente, os itens de 5 a 8. Quanto maior o índice do item da escala maior sua relevância. O limiar de U$ 3.000,00 é obtido a partir da regulação do BACEN no mercado de câmbio e capitais internacionais vigente [27].

5. Ferramenta de detecção de fraudes

Este capítulo sugere uma implementação concreta de uma ferramenta para detecção de fraudes correspondente ao tema do texto. Essa sugestão de implementação encontra-se disponível através do anexo B. Não somente pautamos a modelagem do sistema em si mas de um ambiente de produção com algumas características relevantes no contexto de aplicações distribuídas de alta disponibilidade comuns ao ecossistema de FinTechs [28], que entendemos ser parte significativa do público alvo deste texto. Deixamos claro também que esta ferramenta, até então, não foi implementada apesar de apresentarmos algumas sugestões de implementação ao longo do texto.

Detalhamos aqui as etapas de desenvolvimento sugeridas:

1. Identificação dos padrões transacionais associados às fraudes de interesse.
2. Modelagem de grafos respectivos aos padrões citados no item anterior.
3. Modelagem do banco de dados orientado a grafos.
4. Definição do mapeamento entre os bancos de dados relacional e orientado a grafos.
5. Especificação de consultas em linguagem Cypher para identificar os padrões do item (1) na modelagem do item (3).

6. Estudo de caso - desenvolvimento e implantação de um sistema de monitoramento das fraudes propostas em remessas.
7. Elaboração da análise dos resultados do item (6).
8. Análise comparativa entre o desempenho computacional das consultas SQL vs Cypher em seus respectivos SGBDs e modelagens.

5.1 **Desenvolvimento do Sistema**

O sistema proposto trata-se de um sistema web (cliente-servidor) estabelecido sobre a especificação arquitetural para RESTful-APIs utilizando atentamente os recursos oferecidos pelo protocolo HTTP 1.1 e seguindo a abordagem de DDD (*domain driven design*) a fim de permitir futuras expansões de complexidade sistêmica com baixo custo de manutenção da engenharia do software. Seu desenvolvimento será integralmente em JavaScript rodando sobre a Node.js® runtime. Para persistência de dados, será utilizada uma abordagem poliglota combinando banco de dados relacional e banco de dados orientado a grafos, conforme os respectivos SGBDs: MariaDB e Neo4J. A infraestrutura será fundamentada no serviços de nuvem da Amazon (*Amazon Web Services* - AWS) e containers (Docker), utilizando mecanismos de CI/CD (*continuous integration and deployment*) e controle de versões a serem definidos a posteriori todavia que permitam uma fácil reprodutibilidade do ambiente de execução.

Todas ferramentas a serem utilizadas na composição do sistema e em sua implantação possuem licenças de software permissivas a fins acadêmicos e priorizamos o uso de softwares livres.

5.2 **Arquitetura do Sistema Proposto**

Trabalharemos com persistência em dois tipos de SGBDs e três modelagens. Primeiramente as modelagens abordadas aqui visam contemplar completamente as necessidades de ambos bancos de dados dentro do escopo do trabalho. Além da modelagem dos bancos de dados, trazemos também a modelagem dos fluxos de dados dos *endpoints* da API do sistema. Com estes, por sua vez, temos a intenção de determinar uma especificação em alto nível do comportamento do sistema. Não obstante, ficará desenhado o processo de ETL (*extract, transform, load*) como isomorfismo entre as modelagens.

As modelagens propostas são respectivas a um SGBD Relacional, a um SGBD Orientado a Grafos e a um SGBDOG. Embora não planejamos realizar a persistência da representação em forma de documento, é através desta que ocorre os acessos de Entrada/Saída da API proposta, contemplando também validações de consistência de dados e dos tratamento de erros; o que é imediato da naturalidade da manipulação de dados entre aplicações NodeJS e um esquema de dados orientado a documentos no formato JSON. Ou seja, fica possível definir de forma clara dentro da arquitetura de camadas da API quais são as responsabilidades de cada camada no que diz respeito à manipulação de dados e às regras sistêmicas ou regras de negócio do sistema - o que é um caso claro dos padrões de projeto *Repository* [15] e *Service* [16], por de um lado tratarmos de abstrações em camadas da lógica de aplicação e do outro da lógica de entrada-e-saída .

As camadas da arquitetura deste sistema implementam o modelo arquitetural “Clean Architecture” [25]

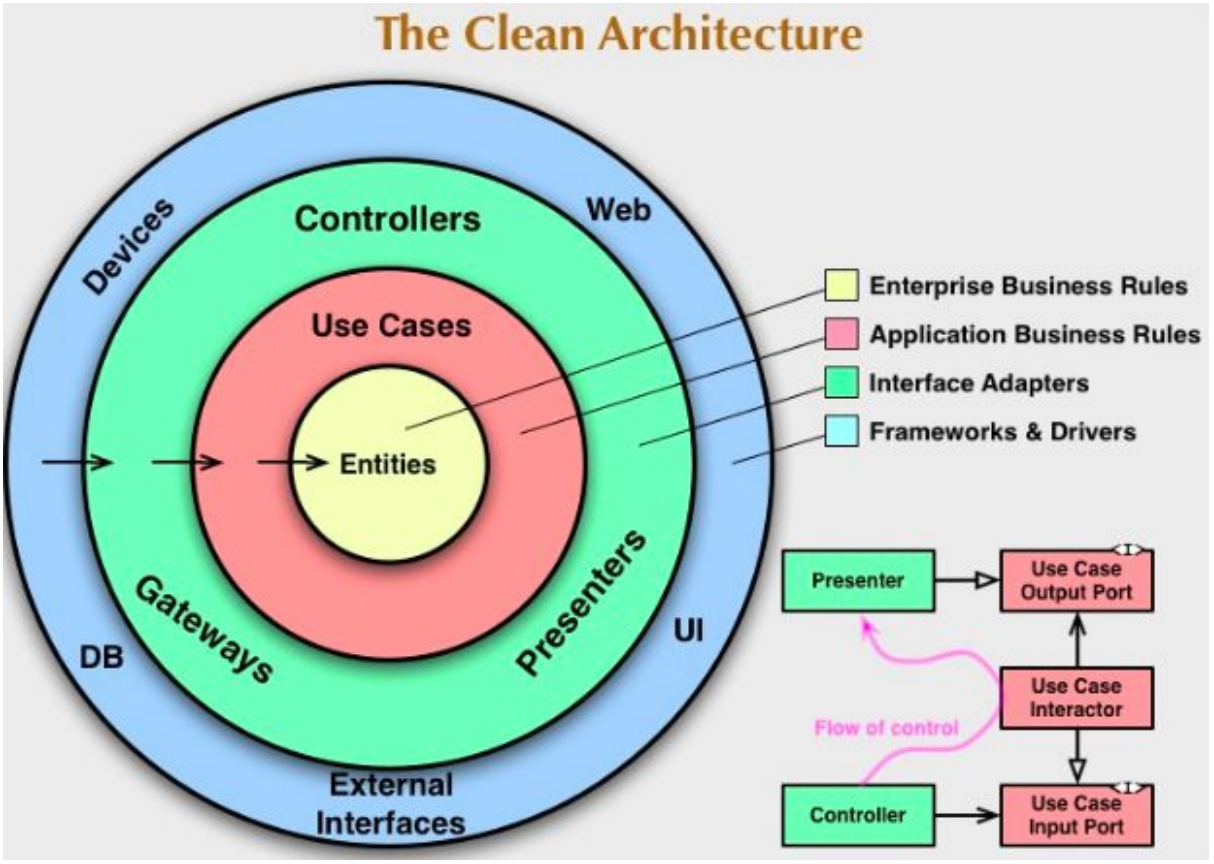

Fig. C. Diagrama de arquitetura de software em camadas, retirado de:
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html, em Setembro de 2019

Onde temos as seguintes correspondências nas camadas:

DB: PostresSQL, Neo4J

Controllers, Gateways e Presenters: NodeJS, KOA Framework

Use Cases: relatórios

Entities: Implementação das heurísticas de pontuação, manipulação dos padrões de suspeitas de fraudes.

5.3 **Modelagem Orientada a Documento**

Esta é a modelagem referente ao corpo da requisição do *endpoint* POST da API, através desta é possível executar regras de validações e análises sobre a entrada, na aplicação. Como por exemplo correlacionar moedas e valores para definir a criticidade das análises a serem executadas. Também é esta representação intermediária entre as outras duas modelagens (ver seção sobre Processo ETL). A camada de Service da API deve, quando se referir à entidade *Transactions* (definida na figura 4, abaixo) - a versão relacional da entidade encontra-se na figura 1, seção 3.2.1, passar como argumento um documento como este através de sua camada de acesso a dados (Repository).

```
{
  title: 'Transaction',
  properties: {
    id: {
      type: 'string',
      encrypt: {
        key: cypterKeys.transaction
      },
      excludeFromView: true
    },
    amount: {
      type: 'decimal',
    },
    currency: {
      type: 'string',
      isRequired: true,
      maxLength: 3,
      minLength: 3
    },
    date: {
      type: 'string',
      isRequired: true
    },
    purposeId: {
      type: 'integer',
      isRequired: true
    },
    purposeName: {
      type: 'string',
      excludeFromView: true,
      join: {
        field: 'name',
        collection: tables.purpose,
        on: {
          from: 'id',
          to: 'purposeId'
        }
      }
    },
    customer: {
      type: 'object',
      required: true,
      title: 'Customer',
      properties: {
```

```
45.         documentNumber: {
46.           type: 'string',
47.           isRequired: true,
48.         },
49.         name: {
50.           type: 'string',
51.           isRequired: true,
52.         },
53.         nature: {
54.           type: 'string',
55.           isRequired: true,
56.           enum: ['l', 'n']
57.         },
58.         country: {
59.           type: 'string',
60.           isRequired: true,
61.           maxLength: 3,
62.           minLength: 3
63.         },
64.         addressLine: {
65.           type: 'string'
66.         }
67.       }
68.     },
69.     beneficiary: {
70.       type: 'object',
71.       required: true,
72.       title: 'Beneficiary',
73.       properties: {
74.         bankAccountCode: {
75.           type: 'string',
76.           isRequired: true,
77.         },
78.         name: {
79.           type: 'string',
80.           isRequired: true,
81.         },
82.         country: {
83.           type: 'string',
84.           isRequired: true,
85.           maxLength: 3,
86.           minLength: 3
87.         },
88.         customerDocumentNumber: {
89.           type: 'string'
90.         },
91.       },
92.     }
93.   }
```

fig 4. JSON Schema da entidade Transaction

5.4 Endpoints

Teremos dois *endpoints* principais na API da ferramenta proposta, um para criação/inserção de novas transações (via requisições HTTP POST) e um para consulta de transações e seus relatórios de análise (via requisições HTTP GET). Outros *endpoints* secundários podem vir a ser adicionados, entretanto estes seriam utilitários à aplicação ou

serviriam como atalhos a casos particulares destes dois. A documentação a respeito da entrada e saída de dados para todos os *endpoints* estará disponível na documentação da API.

5.4.1 POST Transaction Endpoint

```
Fluxo de processamento:
   0.Validação de dados segundo JSON Schema
   1. ETL de entrada
   2. Obter regras de tratativa a serem aplicadas sobre a
      Transação recém adicionada
   3. Executar tratativas aplicáveis à transação e salvar seus
      resultados (2.5.3)
   4. Calcular e salvar escores de risco da Transação, Cliente
      e Beneficiário (2.5.4)
```

Seja uma instância de um ORM JSON-SQL, “orm”, cuja utilização é empregada nos acessos ao SGBDR nos seguintes passos.

1. verificar se o valor de “transaction.customer.document_number” existe na tabela “customer”

   1.1 Caso exista: defina a variável “customerDocumentNumber” com seu valor

   1.2 Caso contrário: insira “transaction.customer” na tabela “customer” retornando a coluna “document_number” e defina a variável “customerDocumentNumber” com este valor

2. verificar se o valor de “transaction.beneficiary.document_number existe” na tabela “beneficiary”

   2.1 Caso exista: defina a variável “beneficiaryBankAccountCode” com seu valor

2.2 Caso contrário: insira “transaction.beneficiary” na tabela “beneficiary” retornando a coluna “bank_account_code” e defina a variável “beneficiaryBankAccountCode” com este valor

3. Verificar se a natureza da transação (transaction.purpose_id) corresponde à natureza “disponibilidade no exterior”

3.1: Se sim: atualizar a coluna “customer_document_number” da tabela “beneficiary” com o valor definido em “customerDocumentNumber” para o beneficiário determinado por “beneficiaryBankAccountCode”

4. Remover referências a colunas não presentes na tabela “transaction” do JSON “transaction”

4.1 Eliminar referência “transaction.customer”

4.2 Eliminar referência “transaction.beneficiary”

5. Inserir “transaction” em sua respectiva tabela

Abaixo deixamos uma sugestão de implementação na linguagem de programação JavaScript versão ES6, usando Knex [18] como ORM; e, um filtrador de campos, “modelPresenter”, que certifica que nenhum dado indevido está sendo utilizado como parâmetro na construção das consultas.

```
async create(data) {
  const { knex, modelPresenter } = this;
  const customer = modelPresenter.in(Object.assign({}, data.customer));
  const beneficiary = modelPresenter.in(Object.assign({}, data.beneficiary));

  let customerDocumentNumber = await knex
    .select('document_number')
    .from(tables.customer)
    .then((rows = []) => rows[0]);

  let beneficiaryBankAccountCode = await knex
    .select('bank_account_code')
    .from(tables.beneficiary)
    .then((rows = []) => rows[0]);

  if (!customerDocumentNumber) {
    customerDocumentNumber = await knex(tables.customer)
      .insert(customer, ['document_number'])
      .then((rows = []) => rows[0].document_number);
  }

  if (!beneficiaryBankAccountCode) {
    beneficiaryBankAccountCode = await knex(tables.customer)
      .insert(beneficiary, ['bank_account_code'])
      .then((rows = []) => rows[0].bank_account_code);
  }

  if (data.purpose_id === ABROAD_AVAIABILITY) {
    knex(tables.beneficiary)
      .where('bank_account_code', beneficiaryBankAccountCode)
      .update('customer_document_number', customerDocumentNumber);
  }

  const transaction = modelPresenter.in(Object.assign({}, data, {
    customerDocumentNumber,
    beneficiaryBankAccountCode
  }));

  delete transaction.customer;
  delete transaction.beneficiary;

  await knex(tables.transaction)
    .insert(transaction)
    .then((rows = []) => rows[0]);
```

Fig.5 sugestão de implementação do ETL de entrada em JavaScript ES6 e usando Knex como ORM. Disponível no anexo B

ETL JSON-OG

0. Pré-condições

0.1 Seja “transaction” como na fig.4

0.2 Estejam definidas restrições de unicidade para “customer.documentNumber” e “beneficiary.bankAccountCode”

0.3 Exista uma conexão ativa com BD

0.4 Esteja definido uma instância de um **OGM** (Object-Graph Mapping) JSON-Cypher, “ogm”, cuja utilização é empregada nos acessos ao SGBDOG nos seguintes passos

1. Garantir a existência de um nó “customer” com a propriedade “documentNumber” como no dado de entrada “transaction.customer.documentNumber”

2. Garantir a existência de um nó “beneficiary” com a propriedade “bankAccountCode” como no dado de entrada “transaction.beneficiary.bankAccountCode”

3. Criar um novo relacionamento (aresta) entre os dois nós criados acima, contendo as propriedades simples do object “transaction” de entrada e com orientação de “customer” para “beneficiary”

Utilizando Neode [11] como OGM a interação com o Neo4J é imediata, primeiramente definimos um modelo representativo de cada nó em JSON e em seguida realizamos chamadas da API do Neode.

```
const customerOGM = {
  document_number: {
    primary: true,
    type: 'string',
    required: true
  },
  name: {
    type: 'string',
    required: true
  },
  country: {
    type: 'string',
    required: true
  },
  nature: {
    type: 'string',
    required: true
  },
  date_of_birth: {
    type: 'string',
    required: true
  },
  address_line: {
    type: 'string',
    required: true
  },
  sends: {
    type: 'relationship',
    target: 'Beneficiary',
    relationship: 'transaction',
    direction: 'out',
    properties: {
      amount: 'decimal',
      currency: 'string',
      date: 'string',
      purpose: 'string',
      rdmsId: 'string'
    }
  }
};

module.exports = customerOGM;
```

Fig.6 Representação Neode do modelo para o nó “customer” do SGBDOG. Disponível no anexo B.

Note que para facilidade de mapeamento das duas representações da entidade Transação em cada um dos SGBDs, adicionamos a propriedade “rdmsId” - definido como o valor da chave primária da tupla respectiva no SGBDR. Sua utilização é opcional embora possa servir como recurso de otimização em possíveis consultas.

Assim sendo, podemos executar as chamadas à API do Neode e inserir as instâncias dos modelos. Devemos executar, para as entidades “customer” e “beneficiary”, chamadas ao método “merge” ao invés do método “create”

```
  const {
    customer: customerOgmInstance,
    beneficiary: beneficiaryOgmInstance,
    transction: transactionOgmInstance
  } = this.buildOgmFromJson({ transaction, customer, beneficiary });

  // CYPHER INSERT
  await neode.mergeOn('customer', { document_number: customerDocumentNumber }, customerOgmInstance);
  await neode.mergeOn('beneficiary', { bank_account_code: beneficiaryBankAccountCode }, beneficiaryOgmInstance);
  neode.create('transaction', transactionOgmInstance);
}
```

Fig.7 Sugestão de uso da API do Neode para realizar a inserção dos dados transacionais. Disponível no anexo B.

O método “merge” primeiramente verifica a existência do nó (através de uma comparação de atributos, caso não exista uma correspondência um nó novo é criado e caso contrário o comando é ignorado. Através do Neode temos disponível o método “mergeOn” que realiza a mesma mecânica de comparação descrita anteriormente para o “merge” no entanto é possível informar dois conjuntos de atributos diferentes para comparação e para inserção, assim fica bastante simples a inserção de um nó usando como predicado de decisão para esta ação apenas o retorno da busca pelo subconjunto de atributos que compõem sua chave primária.

5.4.2 GET Transaction Endpoint

```
Fluxo de processamento:
   0. Validação de filtros e parâmetros
   1. ETL de Saída
```

2. Formatação da apresentação de dados a ser devolvida

O algoritmo referido acima decidirá quando acessar cada um dos bancos de dados. Uma busca por transações com determinados marcadores (*flags*) será endereçada ao banco de dados relacional enquanto buscas por padrões em grafos será endereçada ao banco de dados de grafos. Será possível também realizar consultas que resultarão em uma composição de consultas aos dois bancos de dados.

5.5 Processos ETL

No contexto da ferramenta de detecção de suspeitas de fraudes proposta, temos dois processos ETL. O primeiro para entrada de transações e outro para consultar as transações.

5.5.1 ETL de Entrada

O propósito deste processo é garantir o armazenamento tanto segundo a modelagem relacional (vide figura 1) quanto orientada a grafos (vide seção 3.2.2) a partir da representação JSON de entrada (vide figura 4).

Para tanto, primeiramente devemos verificarmos a pré-existência de *customers* e *beneficiaries* associados no objeto *transaction* na base de dados relacional a fim de garantirmos a existência destes, seja através de uma ação nula ou de inserções. Seguido disso atualizar os dados bancários do beneficiário caso a natureza da operação seja Disponibilidade no Exterior (*own account abroad*) a fim de mantermos atualizado os dados dos clientes (fica aqui a sugestão de monitorar trocas regulares de contas do cliente e manter um histórico de contas utilizadas). Isto feito, podemos inserir o restante da transação, ou seja, seus dados excluindo o que diz respeito a dados de clientes e beneficiários.

Como a parte de *load* do processo de ETL de entrada é associativa entre o SGBDR e o SGBDOG por não haver relação entre cada um deles, podemos definir arbitrariamente que a inserção relacional (descrita acima) é executada primeiro e na sequência temos a respectiva orientada a grafos.

Para realizarmos a inserção no SGBDOG partindo da entrada JSON (vide figura 4), é relativamente mais simples quando comparamos com o processo relacional. Basta acionarmos ações de *merge* para os nós de *customer* e *beneficiary* e *create* para *transaction*.

Abaixo ilustramos (figura 12) um exemplo deste processo, codificado no esboço de implementação disponibilizado (Anexo B).

```
const { tables, purposes } = require('../../../../common/enum-helper');

class TransactionRepository {
  constructor({ knexInstance }) {
    this.knex = knexInstance;
    this.modelPresenter = this.modelPresenter;
  }

  async create(data) {
    const { knex, neode, modelPresenter } = this;
    const customer = modelPresenter.in(Object.assign({}, data.customer));
    const beneficiary = modelPresenter.in(Object.assign({}, data.beneficiary));

    let customerDocumentNumber = await knex
      .select('document_number')
      .from(tables.customer)
      .then((rows = []) => rows[0]);

    let beneficiaryBankAccountCode = await knex
      .select('bank_account_code')
      .from(tables.beneficiary)
      .then((rows = []) => rows[0]);

    if (!customerDocumentNumber) {
      customerDocumentNumber = await knex(tables.customer)
        .insert(customer, ['document_number'])
        .then((rows = []) => rows[0].document_number);
    }

    if (!beneficiaryBankAccountCode) {
      beneficiaryBankAccountCode = await knex(tables.customer)
        .insert(beneficiary, ['bank_account_code'])
        .then((rows = []) => rows[0].bank_account_code);
    }

    if (data.purpose_id === purposes.OWN_ACCOUNT_ABROAD) {
      knex(tables.beneficiary)
        .where('bank_account_code', beneficiaryBankAccountCode)
        .update('customer_document_number', customerDocumentNumber);
    }

    const transaction = modelPresenter.in(Object.assign({}, data, {
      customerDocumentNumber,
      beneficiaryBankAccountCode
    }));

    delete transaction.customer;
    delete transaction.beneficiary;

    await knex(tables.transaction)
      .insert(transaction)
      .then((rows = []) => rows[0]);

    const {
      customer: customerOgmInstance,
      beneficiary: beneficiaryOgmInstance,
      transction: transactionOgmInstance
    } = this.buildOgmFromJson({ transaction, customer, beneficiary });

    // CYPHER INSERT
    await neode.mergeOn('customer', { document_number: customerDocumentNumber }, customerOgmInstance);
    await neode.mergeOn('beneficiary', { bank_account_code: beneficiaryBankAccountCode }, beneficiaryOgmInstance);
    neode.create('transaction', transactionOgmInstance);
  }
}
```

Figura 12 - Processo de ETL de entrada para criação de uma transação. Disponível no anexo B.

5.5.2 ETL de Saída

O processo de saída é menos complexo que o de entrada. Podemos utilizar um ORM definido sobre o modelo da figura 4 e realizarmos consultas através dele. No entanto, pode ser interessante a depender da finalidade da consulta, também consultarmos os dados referentes as resultados das análises, tais como as *flags* e *scores* em suas respectivas tabelas e anexarmos seus respectivos objetos no objeto de saída.

6. Consultas sobre Modelagem

Mostraremos aqui um exemplo teórico de padrão transacional inspirado em um cenário prático. Este exemplo contempla três cenários diferentes de possibilidade de fraude.

No Anexo A, temos as entradas para construção de uma base de dados que contém o grafo da figura 8. O qual é a base de dados sobre a qual as consultas seguintes são executadas.

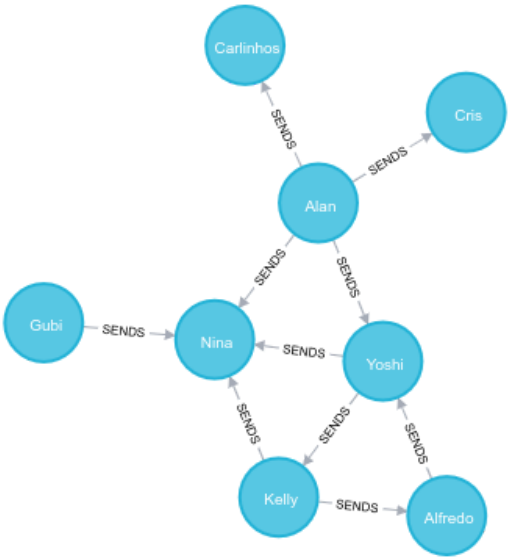

Fig. 8 Grafo de transações ilustrando padrões de fraude

O seguintes grafo é o resultado da consulta (*consulta 3*) que busca um beneficiário com muitos (pelo menos três) remetentes - fixados os parâmetros como valores, datas e naturezas. Devemos interpretar esse resultado da seguinte maneira: o nó cuja aresta se dirige para ele é respectivo a um cliente envolvido em um padrão suspeito. Em uma investigação, caberia verificar, através de uma segunda consulta, outros nós conectados a este.

Fig. 9 Grafo de transações ilustrando resultado de uma busca muitos para um sobre o grafo da figura 8

O seguintes grafo é o resultado da consulta (*consulta 2*) que busca remetentes com múltiplos beneficiários (pelo menos dois) - fixados os parâmetros como valores, datas e naturezas. Devemos interpretar esse resultado da seguinte maneira: todos os nós retornados estão envolvidos no padrão citado. Em uma investigação, caberia verificar, através de uma segunda consulta, outros nós conectados a estes.

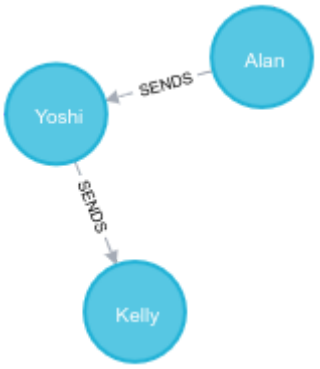

Fig. 10 Grafo de transações ilustrando resultado de uma busca um para muitos sobre o grafo da figura 8

O seguintes grafo é o resultado da consulta (*consulta 1*) que busca anéis de fraude com tamanho maior ou igual a três - fixados os parâmetros como valores, datas e naturezas. Devemos interpretar esse resultado da seguinte maneira: todos os nós retornados estão envolvidos no padrão citado. Diferentemente das outras duas consultas, esta não necessitaria de outras consultas para elaborar um análise mais refinada.

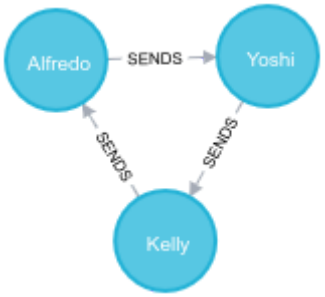

Fig. 11 Grafo de transações ilustrando resultado de uma busca por um padrão circular sobre o grafo da figura 8

7. Considerações Finais

Mostramos ser possível buscar padrões de lavagem de dinheiro e evasão de divisas através da linguagem Cypher de forma mais simples e com melhor desempenho do que uma reprodução de resultados via SQL segundo o cenário teórico desenvolvido. Devido à metodologia do estudo, limitada ao estudo sobre um grafo pequeno com representações construídas, garantimos que a afirmação sobre desempenho se sustenta com base na baixa medida de centralidade dos grafos estudados no contexto abordado. No entanto, acreditamos que em uma implementação do sistema proposto, poderíamos aferir que a proposição sobre desempenho também vale para cenários reais e com uma massa de dados volumosa. Para cenários reais que exijam alto desempenho, medidas de engenharia de infraestrutura além do escopo deste trabalho devem ser implantados - deixamos a recomendação [26] de Ian Robinson abordando esse ponto.

8. Anexos

A)

```
create (Gubi:Customer {name: 'Gubi', documentNumber: '17218695302'})

create (Kelly:Customer:Beneficiary {name: 'Kelly', documentNumber:
'35329921406', bankAccountCode: 'CH5009291485697980788', country:
'CHE'})

create (Alfredo:Customer:Beneficiary {name: 'Alfredo', documentNumber:
'55636331444', bankAccountCode: 'CH8700293001211479221', country:
'CHE'})

create (Alan:Customer {name: 'Alan', documentNumber: '42817926510'})

create (Nina:Beneficiary {name: 'Nina', bankAccountCode:
'CH5604835012345678009', country: 'CHE' })

create (Cris:Beneficiary {name: 'Cris', bankAccountCode:
'CH8469876545747474774', country: 'CHE' })

create (Carlinhos:Beneficiary {name: 'Carlinhos', bankAccountCode:
'CH1943242534953998811', country: 'CHE' })

create (Yoshi:Customer:Beneficiary {name: 'Yoshi', documentNumber:
'84438276705', bankAccountCode: 'CH1100293001211479231', country: 'CHE'
})

MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Gubi' AND b.name = 'Nina'

CREATE (c)-[r:SENDS { amount: 32500, currency: 'USD', purpose:
'Resident Maintenance', date: '25/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Kelly' AND b.name = 'Nina'

CREATE (c)-[r:SENDS { amount: 14750, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Kelly' AND b.name = 'Alfredo'

CREATE (c)-[r:SENDS { amount: 10250, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Alfredo' AND b.name = 'Yoshi'
```

```
CREATE (c)-[r:SENDS  { amount: 10250, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Yoshi' AND b.name = 'Kelly'

CREATE (c)-[r:SENDS { amount: 10250, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Yoshi' AND b.name = 'Nina'

CREATE (c)-[r:SENDS { amount: 12500, currency: 'USD', purpose:
'Resident Maintenance', date: '27/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Nina' AND b.name = 'Yoshi'

CREATE (c)-[r:SENDS { amount: 12500, currency: 'USD', purpose:
'Resident Maintenance', date: '27/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Alan' AND b.name = 'Nina'

CREATE (c)-[r:SENDS { amount: 180500, currency: 'USD', purpose:
'Services', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Alan' AND b.name = 'Cris'

CREATE (c)-[r:SENDS { amount: 700375, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Alan' AND b.name = 'Carlinhos'

CREATE (c)-[r:SENDS { amount: 700375, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)


MATCH (c:Customer),(b:Beneficiary)

WHERE c.name = 'Alan' AND b.name = 'Yoshi'
```

```
CREATE (c)-[r:SENDS { amount: 700375, currency: 'USD', purpose:
'Resident Maintenance', date: '26/01/19' }]->(b)
```

B) Códigos da implementação da ferramenta de detecção de suspeita de fraudes suerida

Os códigos do sistema se encontram disponíveis em: https://github.com/vfaria/stam

9. **Glossário Técnico**

- AWS - Plataforma de Computação em Nuvem da Amazon Inc.
- CI/CD - Metodologias de engenharia de software que automatizam o processo de deploy e guiam um desenvolvimento estável de software sob a ótica de operações [6][7].
- Cypher - Linguagem de consulta declarativa de grafos [8].
- DDD - Padrão arquitetural de software [9]
- Docker - Software para virtualização em nível de sistemas operacionais [10].
- ETL - *Extract Transform Load* - processo de migração entre representações e armazenamento de dados dividido em três etapas
- FinTechs
- HTTP 1.1 - Hypertext Transfer Protocol - https://www.w3.org/Protocols/rfc2616/rfc2616.html.
- HTTP POST - método HTTP - https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5.
- JavaScript - Linguagem interpretada de programação de alto nível
- SGBD - Sistema Gerenciador de Banco de Dados.
- SQL - Linguagem de Consulta Estruturada para bancos de dados relacionais
- Maria DB - SGBD Relacional
- Neo4J - SGBD Orientado a Grafos
- NodeJS - Sistema de Execução JavaScript mantido pela Linux Foundation
- RESTful API - Padrão arquitetural para desenvolvimento de *web services* baseado em Representational State Transfer

- OGM - Object-Graph Mapping / ORM - Object-Relational Mapping - técnica de programação para converter dados entre sistemas de tipos incompatíveis usando linguagens de programação orientadas a objetos. Isso cria, com efeito, um "banco de dados de objetos virtuais" que pode ser usado de dentro da linguagem de programação. OGM é a terminologia para vincular uma base de dados orientado a grafos a um "banco de dados de objetos virtuais" e ORM é seu análogo para bancos de dados relacionais.

10. **Lista de Figuras**

11. **Referências**

[1] BACEN. Resolução n° 2554, de 24 de Setembro de 1998. Dispõe sobre a implantação e implementação de sistema de controles internos.

[2] OLIVEIRA, Luís Flávio Zampronha de. **REMESSAS DE CAPITAIS AO EXTERIOR: a lavagem de dinheiro através da evasão de divisas**. 2012. 72 f. Monografia para obtenção do título de Especialista em Gestão de Política de Segurança Pública da Academia Nacional de Polícia.

[3] Secretaria da Receita Federal do Brasil, Centro de Estudos Tributários e Aduaneiros. **Análise da Arrecadação das Receitas Federais**. Maio de 2018.

[4] RALHA, Cecília et al, **Banco de Dados em Grafo: Um Estudo de Caso em Detecção de Fraudes no Governo Brasileiro**. 2017. Universidade de Brasília.

[5] HOLANDA, Maristela; ARAUJO, Gabriel M. **Uso de banco de dados orientado a grafos na detecção de fraudes nas cotas para exercício da atividade parlamentar**. 2018 Universidade de Brasília.

[6] FOWLER, Martin. **Continuous Integration**, disponível em <https://martinfowler.com/articles/continuousIntegration.html>, acessado em Abril de 2019.

[7] FOWLER, Martin. **Continuous Delivery**, disponível em <https://martinfowler.com/bliki/ContinuousDelivery.html>, acessado em Abril de 2019.

[8] NADIME, Francis; Alastair Green; Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, et al.. **Cypher: An Evolving Query Language for Property Graphs**. SIGMOD'18 Proceedings of the 2018 International Conference on Management of Data. 2018, pp.1433.

[9] EVANS, Eric. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Addison-Wesley. ISBN 978-032-112521-7.

[10] HYKES, Solomon. **Docker (Software)**, disponível em <https://en.wikipedia.org/wiki/Docker_(software)>, acesso em Abril de 2019.

[11] COWLEY, Adam. **Interacting with Neo4j in NodeJS using the Neode Object Mapper**, disponível em <https://medium.com/neo4j/interacting-with-neo4j-in-nodejs-using-the-neode-object-mapper-3d99cb324546>, acesso em Julho 2019.

[12] Neo4J Official Reference, **Concepts: Relational to Graph**, disponível em <https://neo4j.com/developer/graph-db-vs-rdbms/>.

[13] ROCHA, José, **Understanding Neo4j's data on disk**, disponível em <https://neo4j.com/developer/kb/understanding-data-on-disk/>.

[14] COLLIER, Andrew B., **Installing Neo4j on Ubuntu 16.04**, disponível em <https://datawookie.netlify.com/blog/2016/09/installing-neo4j-on-ubuntu-16.04/>

[15] HIEATT, Edward and MEE, Rob. **P of EAA Catalog - Repository**, disponível em <https://martinfowler.com/eaaCatalog/repository.html>, acessado em Setembro de 2019.

[16] STAFFORD, Randy. **P of EAA Catalog - Service Layer**, disponível em <https://martinfowler.com/eaaCatalog/serviceLayer.html>, acessado em Setembro de 2019.

[17] Neo4J Official Reference, **The Degree Centrality algorithm**, disponível em <https://neo4j.com/docs/graph-algorithms/current/algorithms/degree-centrality/>, acessado em Setembro de 2019.

[18] Neo4J Official Reference, **Concepts: Relational to Graph**, disponível em <https://neo4j.com/developer/graph-db-vs-rdbms/> , acessado em Setembro de 2019.

[19] Neo4J Official Reference, **What's a graph database**, disponível em <https://neo4j.com/developer/graph-database/#property-graph> , acessado em Outubro de 2019.

[20] Ramez ELMASRI , Shamkant B. NAVATHE, , Pearson, 2015

[21] Ian ROBINSON, Jim WEBBER, Emil ELFREM, **Graph Databases,** O'Reilly Media; 1 edition, 2013

[22] **The Neo4j Cypher Manual**, v3.5, disponível em <https://neo4j.com/docs/cypher-manual/current/>, acessado em Outubro de 2019.

[23] IBM, ACID properties of transactions, disponível em <https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/product-overview/acid.html>, acessado em Outubro de 2019.

[24] Wikipedia, SQL, disponível em <https://en.wikipedia.org/wiki/SQL>, acessado em Outubro de 2019.

[25] Robert C. MARTIN , **Clean Architecture**, disponível em <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>, acesado em Outubro de 2019.

[26] Ian ROBINSON, **Moving Graphs to Production**, disponível em <https://neo4j.com/blog/graphs-to-production-at-scale/>, acesado em Outubro de 2019.

[27] BACEN. Circular nº 3.605, de 29.06.2012 . Dispõe sobre Regulamento do Mercado de Câmbio e Capitais Internacionais, pg. 2, disponível em <https://www.bcb.gov.br/content/estabilidadefinanceira/Documents/cambiocapitais/normas_cambio/rmcci/RMCCI-1-03.pdf>

[28] GUIABOLSO, Entenda o que é Fintech , disponível em <https://blog.guiabolso.com.br/destaques/entenda-o-que-e-fintech/>, acessado em Janeiro de 2020