Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uso de banco de dados orientado a grafos na detecção de fraudes nas cotas para exercício da atividade parlamentar

Gabriel M. Araujo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Maristela Terto de Holanda

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uso de banco de dados orientado a grafos na detecção de fraudes nas cotas para exercício da atividade parlamentar

Gabriel M. Araujo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof. Dr. Donald Knuth
Stanford University

Dr. Leslie Lamport
Microsoft Research

Prof. Dr. Rodrigo Bonifácio
Coordenador do Bacharelado em Ciência da Computação

Brasília, 7 de setembro de 2018

Dedicatória

Eu dedico esse trabalho aos colegas de curso, que ajudaram a sustentar o sofrimento constante que esse curso proporciona. Dedico também para a galera da central e da CJR, onde aprendi muito mais do que assistindo aulas de cálculo 3.

Agradecimentos

Gostaria de agradecer a todos os meus colegas de curso que me ajudaram nesse período, a todos os professores que se empenharam para ensinar seus alunos e melhorar a cada dia, e principalmente ao Stack Overflow.

Resumo

Este trabalho propõe o uso da tecnologia de banco de dados orientado a grafos para a detecção de fraudes na Cota para o Exercício da Atividade Parlamentar – CEAP. Além do uso do banco de dados orientado a grafos, será feita uma plataforma web para expor informações importantes a população e um estudo sobre o impacto dessas informações na política brasileira. O uso dessas tecnologias facilita muito a manipulação de dados bastante relacionados entre si, tanto em questão de complexidade na consulta, quanto em relação a visualização da informação. A proposta em questão foi validada com um estudo de caso, utilizando os dados abertos da Cota para o Exercício da Atividade Parlamentar da câmara dos deputados. Foi desenvolvido um ETL para extrair os dados e popular o banco, em seguida as consultas foram realizadas para detectar as fraudes e obter informações a respeito dos dados, finalmente foi desenvolvido um sistema web que se comunica via REST com o banco de dados para expor as informações a população de forma mais clara e simples. Para trabalhos futuros, seria interessante o uso de aprendizagem de máquina para obter mais informações valiosas sobre a CEAP.

Palavras-chave: Banco de dados orientado a grafos, fraude, política

Abstract

This work proposes the use of graph oriented databases to detect frauds in Quota for the Exercise of Parliamentary Activity. In addition to the use of the graph oriented databases, a web platform will be developed to expose important information to the population and a study on the impact of this information on Brazilian politics. The use of these technologies, greatly facilitates the manipulation of closely related data, both in terms of query complexity, and information visualization. The proposal in question was validated with a case study, using the open data of the Quota for the Exercise of the Parliamentary Activity of the Chamber of Deputies. It was developed an ETL to extract the data and fill the database, then the queries were made to detect the fraud and to obtain information about the data, finally a web system was developed that communicates via REST with the database to expose the information to the population more clearly and simply. For future work, it would be interesting to use machine learning to obtain more valuable information about CEAP.

Keywords: Graph oriented databases, fraud, politics

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O Brasil é reconhecido mundialmente por seus aspectos culturais, por ser membro do grupo de países BRICS e por ser líder da américa latina. Porém, nosso país também é bastante conhecido por ser um país corrupto, e por possuir um cenário político conturbado. Um estudo feito por Abramo [1] compara as relações entre índices de percepção de corrupção e outros indicadores de alguns países latino americanos, e o Brasil se encontra na 49ª (quadragésima nona) posição em um ranking de corrupção dentre 90 países. Já o estudo feito por Filgueiras [2] mostra que de acordo com a percepção dos brasileiros, a Câmara dos vereadores e a Câmara dos deputados são as instituições com maior presença de corrupção.

Atualmente, operações como a Lava Jato, mostram que o país vive um período político muito sensível, que acabam causando vários problemas nos mais diversos setores do país e principalmente na economia. Dessa forma, o objetivo geral deste trabalho é utilizar a tecnologia de banco de dados orientado a grafos, para auxiliar na detecção de possíveis fraudes em um determinado conjunto de dados. Como foi mencionado acima, a câmara dos deputados está entre as instituições com maior presença de corrupção no país, portanto, o conjunto de dados utilizado neste trabalho é a Cota para o Exercício da Atividade Parlamentar – CEAP (antiga verba indenizatória), que é uma cota única mensal destinada a custear os gastos dos deputados exclusivamente vinculados ao exercício da atividade parlamentar.

O Ato da Mesa nº 43 de 2009, detalha as regras para o uso da CEAP, entretanto um deputado pode realizar algumas transações que não são observadas facilmente pelos responsáveis em fiscalizar essas transações. Por exemplo, o artigo 4, parágrafo 13 do Ato da Mesa nº 43 de 2009, diz: *Não se admitirá a utilização da Cota para ressarcimento de despesas relativas a bens fornecidos ou serviços prestados por empresa ou entidade da qual o proprietário ou detentor de qualquer participação seja o Deputado ou parente seu até terceiro grau.* Dessa forma, o Deputado pode realizar transações que violam essa regra,

sendo inviável verificar as relações de parentesco de cada Deputado em cada transação, especialmente se utilizarem tecnologias inadequadas.

Portanto, justifica-se o uso de um banco de dados orientado a grafo para identificar os relacionamentos envolvendo cada transação de um Deputado. Um banco de dados relacional também consegue resolver esse problema, entretanto, com um custo e complexidade bem maior em relação a um banco de dados orientado a grafo. Isso se deve porque os relacionamentos são evidenciados na estrutura de um grafo de forma muito mais natural e simples, onde cada entidade é representada como um nó do grafo e se relaciona com outras entidades por meio de arestas. Devido a essas particularidades, os bancos de dados em grafo vem ganhando bastante popularidade ultimamente [3], registrando a maior taxa de mudança de popularidade de 2013 até 2017.

1.1 Objetivos

O objetivo geral deste trabalho é utilizar o SGBD OrientDB para evidenciar relacionamentos nas transações dos Deputados que violam o artigo 4, parágrafo 13 do Ato da Mesa nº 43 de 2009, que regula a CEAP. Para essa análise, será feito um sistema web que expõe essas informações a população de forma mais simples e clara. Além de estudar como essas tecnologias impactam a política brasileira.

Os objetivos específicos deste trabalho são:

- Implementar o banco de dados em grafo com os dados da CEAP.
- Executar consultas que evidenciem relacionamentos fraudulentos.
- Identificar vantagens e desvantagens no uso de banco de dados orientado a grafo para a detecção de fraudes em conjuntos de dados genéricos.
- Desenvolver um sistema web para expor informações a respeito dos dados.
- Estudar o impacto dessas tecnologias na política e sociedade brasileira.

1.2 Metodologia

Este trabalho foi dividido em duas partes, a primeira teórica e a segunda prática. Na parte teórica foi realizado um estudo baseado em livros, artigos e páginas da *web* sobre os assuntos relacionados a banco de dados, SGBD orientado a grafo, NoSQL, as leis que regem a CEAP e os impactos dessas iniciativas na politica. Já na parte prática foi desenvolvido um ETL para ler os arquivos que contém os dados das transações e popular o banco, em seguida foram realizadas consultas que buscam evidenciar os relacionamentos

atrelados a cada transação, posteriormente foi desenvolvido um sistema web de forma que forneça as informações claramente a todos. Por fim, foi realizado uma análise dos resultados obtidos e conclusões finais.

1.3 Estrutura do Trabalho

Este trabalho está dividido nos seguintes capítulos:

- Capítulo 2: Introduzo os conceitos relacionados a grafos, necessários para a compreensão de um SGBD orientado a grafo. Forneço uma visão geral de SGBD NoSQL, apontando suas principais características e utilidades. Finalmente explico as características de um SGBD orientado a grafo e sua evolução.
- Capítulo 3: Nesse capítulo apresento os meios utilizados para resolver o problema, e uma explicação mais detalhada de todo o processo de desenvolvimento e como as tecnologias auxiliaram nessa resolução.
- Capítulo 4: Apresento a análise dos resultados obtidos ao realizar as consultas e do sistema desenvolvido para fornecer as informações. Essa análise abrange tanto os resultados das consultas, ou seja, se foi possível identificar uma transação ilícita, quanto o impacto da divulgação desse tipo de informação na política brasileira.
- Capítulo 5: Finalmente, apresento minhas conclusões do trabalho realizado e sugestões para trabalhos futuros relacionados com a área.

Capítulo 2

Fundamentação Teórica

Este capítulo descreve toda a fundamentação teórica por trás das tecnologias utilizadas na implementação do estudo de caso. Inicialmente, explico os conceitos gerais relacionados a teoria de grafos, em seguida, aponto as principais características e utilidades dos SGBDs NoSQL. Posteriormente, aponto as características e particularidades de um SGBD orientado a grafos, junto com uma explicação acerca do SGBD escolhido para o trabalho que é o OrientDB. Por fim, explico um pouco sobre a tecnologia REST utilizada para a comunicação entre o sistema desenvolvido e o OrientDB.

2.1 Teoria de Grafos

Essa seção, aborda a teoria por trás da estrutura de um grafo, necessária para a compreensão do funcionamento de um SGBD orientado a grafos. Qualquer SGBD orientado a grafos utiliza algum dos conceitos aqui descritos para a construção do banco de dados.

2.1.1 Definição de um grafo

A definição formal de um grafo pode ser feita da seguinte forma: Um grafo G é uma tripla ordenada $(V(G), E(G), \psi g)$, que consiste de um conjunto não vazio $V(G)$ de vértices, um conjunto $E(G)$, disjunto do conjunto $V(G)$, de arestas, e uma função de incidência ψg que associa cada aresta de G um par não ordenado (não necessáriamente distinto) de vértices de G. Dessa forma, se e é uma aresta e u e v são vértices, de tal modo que $\psi g(e) = uv$, então, diz-se que e faz a união de u e v; Os vértices u e v são chamados de extremidades de e [4].

A figura 2.1 esclarece a definição fornecida no parágrafo acima:

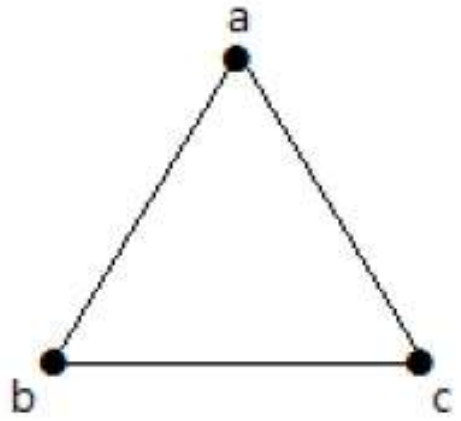

Figura 2.1: Exemplo de uma estrutura de grafo

Portanto, o conjunto $V(G)$ de vértices é não vazio e composto por três vértices, $V(G) = \{a, b, c\}$. Já o conjunto $E(G)$ de arestas é composto por três arestas, $E(G) = \{e1, e2, e3\}$. De forma que a função de incidência ψg é definida da seguinte maneira: $\psi g(e1) = ab$, $\psi g(e2) = ac$ e $\psi g(e3) = bc$. Portanto, a definição formal do grafo acima é $(\{a, b, c\}, \{e1, e2, e3\}, \psi g(e1) = ab, \psi g(e2) = ac\ \psi g(e3) = bc)$.

2.1.2 Propriedades de um grafo

Um grafo possui essa nomenclatura pois pode ser representado graficamente, e a partir dessa representação é possível observar algumas propriedades importantes. Como mostra a figura 2.1, cada vértice é representado por um ponto, e cada aresta é representada por uma linha, juntando dois pontos [4]. Todo grafo possui as propriedades de ordem e tamanho. A ordem de um grafo se refere ao número de vértices, ou seja, a quantidade de elementos no conjunto $V(G)$, enquanto o tamanho de um grafo é referente ao número de arestas, ou seja, a quantidade de elementos no conjunto $E(G)$.

Boa parte da teoria de grafos é baseada em uma categoria específica que é a categoria de grafos simples. Antes de definir um grafo como simples, é necessário definir o que é um laço em um grafo. Intuitivamente, um laço é quando a aresta sai de um vértice e termina no próprio vértice. A figura 2.2 exemplifica um laço em grafo:

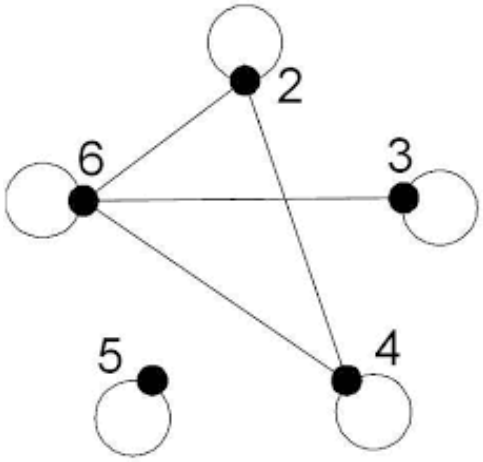

Figura 2.2: Exemplo de uma estrutura de grafo com presença de laço

Dessa forma, um grafo é definido como simples se não possui laços nem duas ou mais arestas ligando dois vértices [4]. Portanto, o grafo 2.2 não é um grafo simples, enquanto o grafo 2.1 é um grafo simples. Uma propriedade básica de um vértice é o seu grau. O grau de um vértice é a quantidade de aretas incidentes nele. Por exemplo, no grafo 2.1 o vértice a possui grau 2.

O grafo 2.1, mostra uma categoria de grafos que não faz distinção entre a orientação das arestas. Um grafo que evidencia a orientação das arestas é dito grafo orientado, como mostra a figura a seguir:

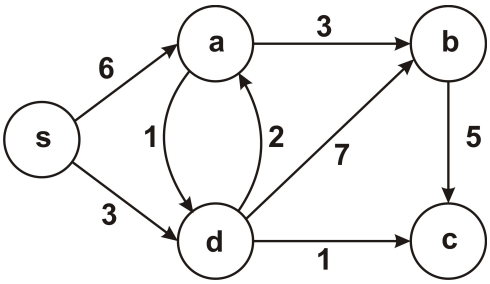

Figura 2.3: Exemplo de uma estrutura de grafo direcionado

O OrientDB utiliza grafos orientados para representar os dados. Alguns relacionamentos exigem que a direção da aresta aponte para os dois lados, o OrientDB representa essa característica com somente uma aresta e uma seta em cada extremidade, mas é importante ressaltar que o correto é representar com duas arestas, uma em cada sentido, como mostra a figura 2.3 nas arestas 1 e 2.

Dentro do universo de teoria de grafos, existem ainda diversas outras propriedades tais como: Isomorfismo entre grafos, subgrafos, caminho, caminho fechado e etc. Essas propriedades fornecem a base para a compreensão e resolução de problemas importantes para a computação, como o problema de menor caminho, que busca encontrar o menor

caminho entre dois vértices em um grafo com pesos. O SGBD utilizado nesse trabalho o OrientDB, fornece uma solução para o problema de menor caminho, e conhecer a teoria por trás do problema faz com que o usuário possa usar da melhor maneira essa solução [5].

2.1.3 Grafos em SGBD NoSQL

2.2 SGBD NoSQL

Essa seção aborda a categoria de bancos de dados não relacionais. Inicialmente, realizo uma introdução sobre o assunto, para em seguida explicitar as principais características e categorias dentro desse grupo de bancos de dados.

2.2.1 Introdução

O termo NoSQL foi utilizado na literatura pela primeira vez por Carlo Strozzi em 1998 como o nome de um banco de dados que ele estava desenvolvendo na época [6]. Curiosamente, é um banco relacional que não possuia interface SQL, portanto, NoSQL. Entre o período dos anos de 2000 e 2005 o universo dos SGBD NoSQL começou a aumentar bastante, o SGBD baseado em grafo Neo4j começou a ser desenvolvido no ano de 2000, o SGBD da Google BigTable[7] começa em 2004 e em seguida o CouchDB se inicia por volta de 2005. Porém, todas essas tecnologias não eram na época categorizadas como NoSQL, somente em 2009 o termo foi reintroduzido por Johan Oskarsson ao organizar um evento para discutir sobre banco de dados não relacionais, distribuídos e de código aberto, e a partir desse momento o termo passa a ser usado para categorizar os bancos de dados não relacionais.

O termo NoSQL gera muita confusão pois leva a entender que os bancos de dados nessa categoria não utilizam a linguagem SQL. Porém, o termo costuma ser utilizado como *Not only SQL*, e demonstra que tais sistemas podem suportar também linguagens de consultas baseadas em SQL. O aspecto mais importante por trás dos SGBD NoSQL é o motivo de seus surgimentos. No início dos anos 2000, a forma como os usuários começaram a interagir na internet começou a evoluir e consequentemente os sistemas e aplicações passaram a gerar e consumir cada vez mais dados e informações. Esse crescimento na geração e consumo de dados foi enorme, e por isso os especialistas necessitavam de tecnologias que atendessem melhor suas necessidades, desse cenário começaram a nascer os primeiros SGBD NoSQL como o BigTable da google e o Dynamo da Amazon[8].

2.2.2 Características de um SGBD NoSQL

Como foi mencionado acima, por volta dos anos 2000 começou a nascer a necessidade de novas tecnologias na área de banco de dados. Os sistemas precisavam de boa escalabilidade horizontal para operações de forma distribuída, entre outras coisas. Vários aspectos relacionados a categoria dos SGBD NoSQL são ainda abertos e não possuem um consenso. O trabalho feito por Rick Cattell[9] aglomerou algumas características que geralmente se encontram nesse tipo de banco de dados:

- Habilidade de escalar horizontalmente operações simples através de vários servidores
- Habilidade de replicar e distribuir os dados através de vários servidores
- Um modelo de concorrência mais flexível que as transações ACID presentes em sistemas relacionais.
- Uso eficiente de índices distribuidos e da memória RAM para armazenar os dados
- Habilidade de adicionar dinamicamente novos atributos a um registro

Porém, é importante ressaltar que nem todas essas características precisam ser atendidas para fazer parte dessa categoria. Sendo que também existem outras características importantes observadas em alguns sistemas como por exemplo, disponibilidade e consistência dos dados, níveis de flexibilidade para o esquema do banco de dados e etc.

2.2.3 Gerenciamento de Transações

Uma propriedade importante em qualquer SBGD é o controle de concorrência. Normalmente, vários usuários solicitam operações ao SGBD simultâneamente, sendo que esse sistema gerenciador de banco de dados precisa garantir aos usuários a integridades dos dados transacionados. A maioria dos bancos de dados relacionais utilizam transações ACID[10] para controlar essa concorrência, que impõe ao SGBD as seguintes propriedades:

- Atomicidade: Essa propriedade impõe que durante uma transação, todas as alterações feitas no banco de dados sejam efetivadas. Caso ocorra algum erro durante a transação o SGBD realiza um *rollback* para o estado consistente anterior a transação.
- Consistência: Essa propriedade impõe que ao término de uma transação o banco de dados está de forma consistente. Ou seja, a cada transação os dados devem estar consistentes e a garantia de que as restrições impostas aos dados não sejam violadas.

- Isolamento: A propriedade de isolamento de uma transação, busca impor ao SGBD que cada transação seja independente das demais. Ou seja, garante que cada transação seja vista como uma unidade e impede que outras transações acessem dados que possam levar o bancos de dados a um estado inconsistente.
- Durabilidade: A durabilidade garante que as informações salvas após cada transação permaneçam no banco de dados. Os dados não podem desaparecer em nenhum momento e devem permanecer persistidos independente de qualquer falha.

Esse formato é utilizado por bancos de dados relacionais, e atende bem uma boa parte de sistemas e aplicações nos dias de hoje, como é o caso de sistemas bancários por exemplo, que necessitam dessas propriedades para garantir a integridades das informações de seus clientes.

O mais importante a se perceber aqui é que nem toda aplicação precisa necessariamente desse controle de concorrência elaborado. No caso de bancos é de suma importância esse tipo de controle, agora em uma aplicação de alta disponibilidade uma propriedade como a consistência pode ser um pouco mais fraca. Portanto, para atender a essas novas demandas, nasceu o modelo de transações BASE[11] que busca fornecer maior flexibilidade para esses novos sistemas. O BASE tem como característica em vez de exigir consistência após cada transação, é suficiente para o banco de dados estar eventualmente consistente. Existe um teorema famoso feito por Eric Brewer conhecido como Teorema CAP[12], que demonstra o trade-off entre consistência, disponibilidade e tolerância a particionamento. Eric diz que se for necessário essas três características, é necessário escolher somente duas delas. Portanto no modelo BASE é possíve escolher tolerância a particionamento e disponibilidade e abrir mão de uma alta consistência dos dados. O acrônimo BASE vem das três seguintes características:

- *Basic Availability*: Essa propriedade no fundo quer dizer que o banco de dados aparenta funcionar a maior parte do tempo.
- *Soft state*: Essa propriedade quer dizer que as bases não precisam estar sempre consistentes, nem as réplicas precisam estar consistentes o tempo inteiro. Ou seja, o estado do sistema é volátil.
- *Eventual consistency*: Essa propriedade diz que as bases de dados vão ficar eventualmente consistentes no futuro.

2.2.4 Categorias de NoSQL

Dentro do universo dos SGBD NoSQL, existem cinco grandes categorias que separam cada SGBD em relação a estratégia de armazenamento[13]. Tais categorias são:

SGBD NoSQL por chave/valor

Os SGBD dessa categoria são de certa forma bem simples, mas bastante eficientes. O dado armazenado é composto por uma *string* que representa a chave, e o dado de fato a ser referenciado pela chave que é o valor, portanto, se cria um par chave/valor [13][14].

Os SGBD NoSQL por chave e valor, costuma valorizar uma maior escalabilidade em troca de consistência, tal característica é mecionada na seção 2.2.3. O seu uso é bastante eficiente para situações como guardar seções de usuário, ou gerenciar um carrinho de compras online [13]. Um exemplo de SGBD dessa categoria é o DynamoDB [8]. A figura abaixo mostra o formato de um dado nesse SGBD em contraste com o formato de SGBD relacional e de SGBD orientado a documentos.

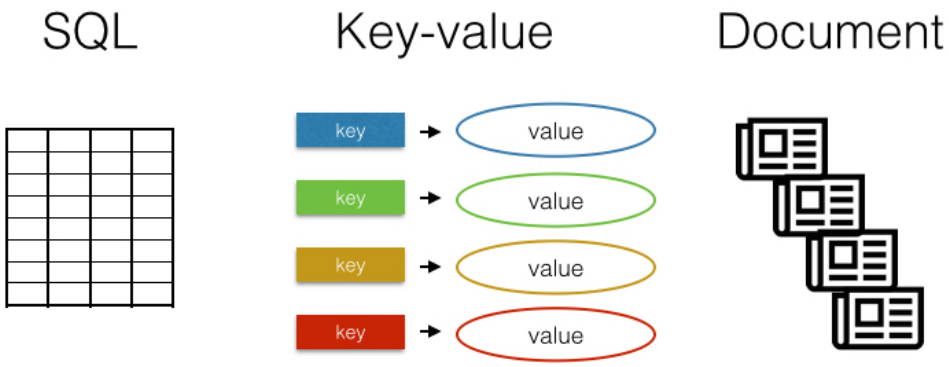

Figura 2.4: Exemplos de estrura relacional, chave/valor e orientada a documentos

SGBD NoSQL orientado por colunas

Os SGBD dessa categoria, podem ser confundidos de certa forma com um SGBD relacional por serem orientado a colunas. Mas na verdade um SGBD NoSQL orientado por colunas tem um formato híbrido entre linha e coluna. Apesar de compartilharem do conceito de armazenamento por colunas, esses SGBD não armazenam os dados em tabelas, e sim em grandes arquitetura distribuídas. Nessa categoria, cada chave é associada com uma ou mais colunas, chamada de atributos. Esse armazenamento é feito de forma que o dado seja agregado mais rapidamente e com menos processamento de entrada e saída [13].

Ou seja, esses banco de dados contém uma coluna extensível de dados fortemente relacionados, em vez de conjuntos de informação em uma estrutura rígida de tabela com linha e colunas como no modelo relacional[14]. Esse tipo de NoSQL é bastante útil para operações de mineração de dados e aplicações de análise [13]. Dois grandes representantes dessa categoria são o BigTable da google [7] e o Cassandra. A figura abaixo exemplifica a diferença da estrutura orientada a colunas e orientada a documentos:

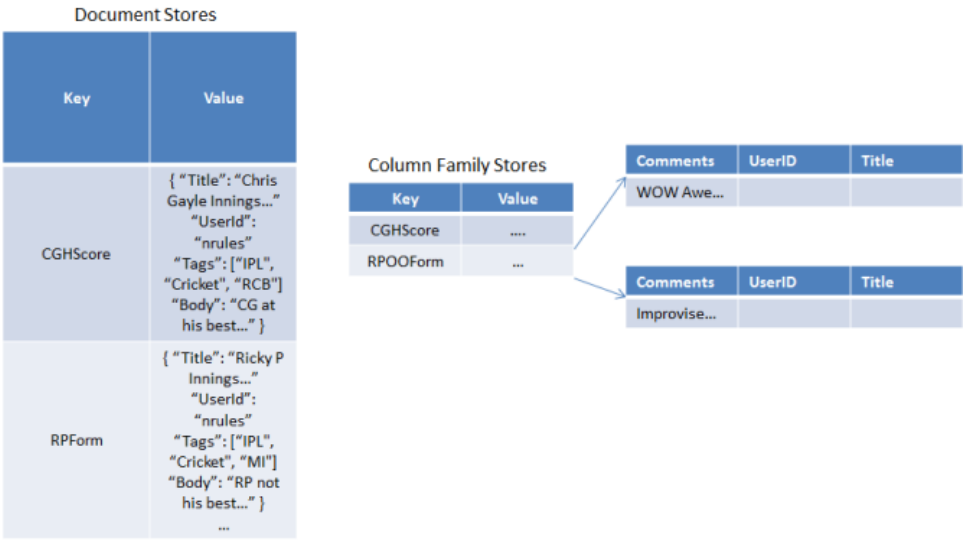

Figura 2.5: Exemplo de estrutura de um NoSQL orientado a colunas.

SGBD NoSQL orientado a documentos

Nessa categoria, os dados são armazenados em forma de documentos. Um documento dentro desse tipo de SGBD pode ser comparado a um registro dentro de um SGBD relacional, mas nos SGBD NoSQL existe uma flexibilidade maior pois é permitido uma estrutura *Schema-less*. Normalmente, esses documentos seguem algum formato padrão como *JSON* ou *XML* por exemplo. Uma diferença com o formato dos bancos relacionais, é que um campo de um registro dentro de um SGBD relacional que não está preenchido, necessariamente ficará vazio, já num SGBD NoSQL orientado a documentos, cada documento pode ter dados similares ou não tão similares, ou seja, um campo sem informação não precisa aparecer como vazio para o usuário, demonstrando a flexibilidade que esses SGBD proporcionam [13].

Cada documento é endereçado por uma chave, da mesma forma que em SGBD NoSQL orientado a chave/valor. Esses SGBD são úteis em aplicações em que os dados não precisam ser armazenados em tabelas com campos uniformes, mas sim armazenados em documentos com características especiais. Normalmente, não é recomendado para dados com muitos relacionamentos ou normalização. Dois representantes dessa categoria são o MongoDB e o CouchDB [13]. A figura 2.5 mostra um exemplo da estrutura de um SGBD orientado a documentos em contraste com um SGBD orientado a colunas.

SGBD NoSQL orientado a grafos

Os SGBD NoSQL orientado a grafos armazenam os dados em uma estrutura de grafo. As características e informações acerca da estrutura de um grafo são abordadas em maiores detalhes na seção 2.1. Resumidamente, um grafo é um conjunto de nós e arestas, em que

os nós são os objetos e as arestas representam o relacionamente entre dois objetos (nós). Esses SGBD usam a técnica conhecida como *index free adjacency*, em que cada nó possui um ponteiro diretamente para nós adjacentes. Essa técnica permite que uma grande quantidade de nós seja percorrida de forma eficiente [13]. A figura abaixo exemplifica o formato de um grafo no SGBD NoSQL OrientDB.

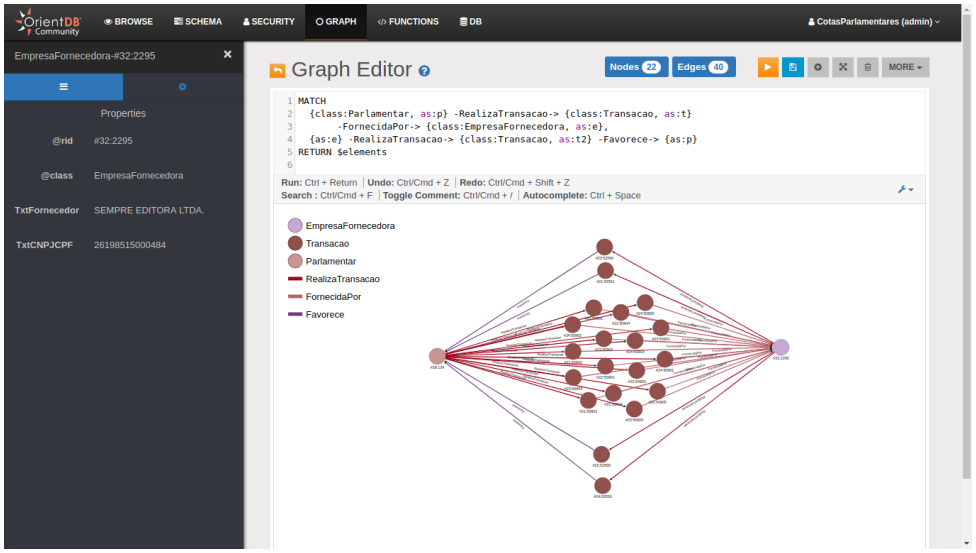

Figura 2.6: Exemplo da estrutura de um grafo no SGBD OrientDB

As consultas são feitas seguindo a ideia de percorrimento do grafo, o que torna esses SGBD mais eficientes que SGBD relacionais. O seu uso varia de aplicações para redes sociais, bioinformática, software de recomendação e etc [13]. O principal ponto de seu uso é em dados que são bastante relacionados entre si, pois a estrutura de um grafo expõe naturalmente os relacionamentos entre os objetos. O maior representando dessa categoria de SGBD é o Neo4j seguido pelo OrientDB e AllegroGraph.

SGBD NoSQL orientado a objetos

Um SGBD orientado a objetos, armazena os dados e informações em objetos, similares aos objetos presentes no paradigma de orientação a objetos. Portanto, essa categoria de SGBD pode ser vista como uma combinação entre os princípios de programação orientada a objetos e banco de dados. Esses banco de dados oferecem todas as característica da programação orientada a objetos tais como: Encapsulamento de dados, polimorfismo e herança. É possível fazer um paralelo entre classe, objeto e atributos da classe com tabela, tupla e colunas em uma tupla dos banco de dados relacionais respectivamente [13].

Cada objeto nessa estrutura possui um identificador único que o representa. O acesso nessa estrutura é mais rápido pois cada objeto pode ser acessado através de ponteiros. O seu uso auxilia no processo de desenvolvimento de software, sendo útil também em aplica-

ções envolvendo relacionamentos complexos entre objetos. Esse tipo de SGBD vem sendo usando em pesquisas científicas, telecomunicações e outras aplicações. Essa categoria tem a desvantagem de estar associada a um tipo específico de linguagem de programação e por possuir dificuldades de escalabilidade, uma vez que o tamanho da memória física é excedido [13]. Um grande representante dessa categoria é o db4o, o trabalho de Kulshrestha, Sudhanshu and Sachdeva e Shelly [15] faz uma comparação de performance entre SGBD relacionais e o db4o. A figura abaixo exemplifica como um objeto é dividído para ser armazenado em tabelas em uma estrutura relaciona, já na estrutura orientada a objetos, o objeto é armazenado sem essa divisão [16].

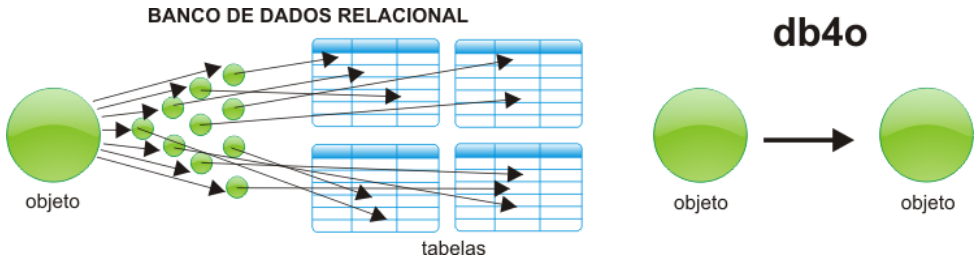

Figura 2.7: Contraste entre estrutura relaciona e orientada a objetos

2.3 Comparação qualitativa entre OrientDB e Neo4j

2.3.1 Introdução

Hoje no mercado existem diversos SGBD orientado a grafos, tais são: OrientDB, Neo4j, AllegroGraph, FlockDB entre outros. Dentre esses, sem dúvida o mais popular é o SGBD NoSQL neo4j. Como foi mencionado no capítulo 1, os SGBD orientado a grafos vem obtendo um grande número de novos usuários [3], registrando a maior taxa de mudança de popularidade entre 2013 e 2017. Hoje o Neo4j se encontra na primeira posição segundo o seguinte ranking [17], e por esse motivo o seguinte capítulo realiza uma comparação entre o OrientDB e o Neo4j.

Trabalhos como o de Jouili [18], Kolomivcenko [19] e Kovacs [20], realizam comparações entre os dois SGBD. Nessas comparações são levados em conta aspectos como performance, licensa comercial ou *open source*, linguagem de consulta e etc. Além desses trabalhos, artigos como o de Barmpis [21] e Labute [22] reforçam o fato do crescimento tanto no uso comercial quanto em pesquisas acadêmicas em banco de dados orientado a grafos.

2.3.2 Comparação qualitativa

O trabalho feito por Angles [23] em 2012 compara alguns SGBD orientado a grafos. Foi identificado três características comuns entre todos os modelos que são: O esquema e as instâncias são modelados como grafos, operações orientada a grafos e um conjunto de regras de integridade. Dessa forma, a tabela 2.1 representa uma comparação entre os modelos de dados do OrientDB e Neo4j:

SGBD	OrientDB	Neo4j
Modelo de banco de dados	*multi-model DBMS*	*Graph DBMS*
Modelos de esquema	*Schema-full*, *Schema-less*, *Schema-hybrid*	*schema-free* e *schema-optional*
Tipos de dados customizados	Sim	Não
Modelo Reativo	Sim	Não

Tabela 2.1: Comparação qualitativa entre modelo de dados do OrientDB e Neo4j

Portanto, podemos observar que o OrientDB é um SGBD multi modelo, pois suporta operações com documentos, chave/valor e grafos. Já o Neo4j trabalha somente com o modelo de grafos. Além disso, o OrientDB possui suporte para diferentes modelos de esquema podendo usar um esquema completo, nenhum esquema ou um esquema híbrido para modelar os dados. O Neo4j se utiliza nenhum esquema na modelagem ou um esquema opcional, baseado no conceito de labels. Nesse esquema opcional, os labels são usados na especificação de índices, e para definir restrições no grafo, juntos ambos formam o esquema do grafo no Neo4j [24]. O OrientDB permite que o usuário crie tipo de dados customizados no esquema, basta que a classe da propriedade já exista no banco de dados [25], enquanto no Neo4j isso não é possível. Finalmente, uma característica interessante que o OrientDB fornece é a de modelo reativo. Essa propriedade é chamada de *live query* [26], e é utilizada para construir aplicações em tempo real. Em uma abordagem tradicional, é inviável que o cliente da aplicação fique o tempo inteiro realizando consultas para obter novas atualizações, pois isso gera um grande desperdício de recursos. A imagem 2.8 resume a situação:

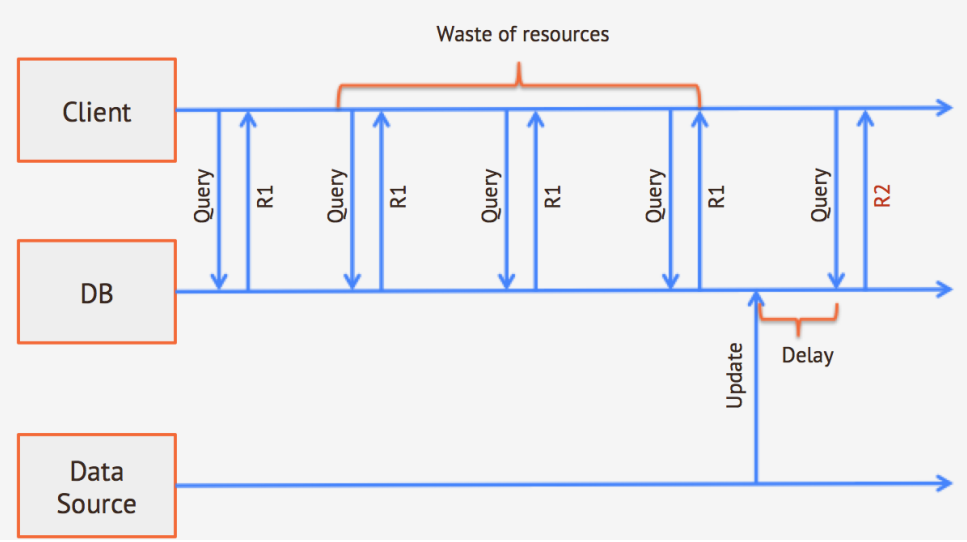

Figura 2.8: Exemplo de tentativa de aplicação reativa

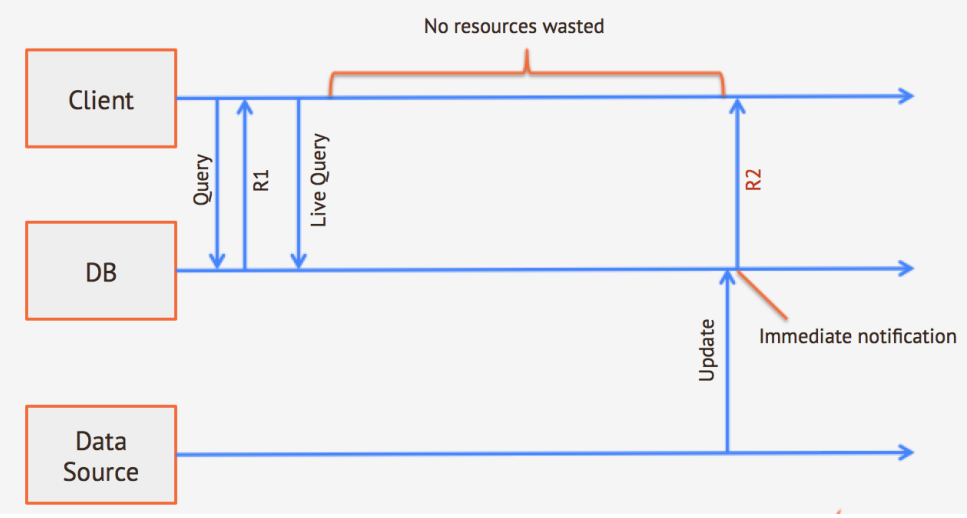

Figura 2.9: Exemplo do uso de *live queries*

Com o uso das *live queries*, esse tipo de problema é resolvido e não há mais desperdício de recursos, como mostra a figura 2.9.

Apesar de várias diferenças, os dois SGBD possuem características em comum. A tabela 2.2 demonstra comparações de características gerais, entre o OrientDB e o Neo4j:

A partir da tabela 2.2, é possível observar algumas características em comum entre os SGBD, por exemplo, ambos trabalham com o modelo de transações ACID, que foi discutido em detalhes na seção 2.2.3. Além disso, os dois suportam o uso de dados e consultas espaciais para trabalhar com dados geográficos, cada um possui uma versão open source, mesmo que sob licensas diferentes. É importante pontuar também que ambos

SGBD	OrientDB	Neo4j
Transações ACID	Sim	Sim
Suporte a dados espaciais	Sim	Sim
Licensa de uso	Apache versão 2	GPL versão 3
Linguagem implementada	Java, Scala	Java
Linguagem de consulta	Linguagem derivada do SQL, sem uso de joins	Cypher
Métodos de particionamento	Sharding	Não
Metodos de replicação	Multi-master replication	Causal Clustering using Raft protocol
Suporte a MapReduce	Sim	Não
Concorrência	Sim	Sim

Tabela 2.2: Comparação qualitativa de características gerais entre OrientDB e Neo4j

possuem suporte a acessos concorrentes e foram desenvolvidos utilizando a linguagem de programação Java.

Uma diferença importante entre os dois SGBD é em relaçao a linguagem de consulta utilizada. O OrientDB possui uma linguagem de consulta derivada da linguagem SQL, isso é bastante vantajoso para aqueles que vem do universo de SGBD relacional, enquanto o Neo4j utiliza uma linguagem de consulta própria conhecida como cypher. O código abaixo exemplifica uma consulta no OrientDB e sua semelhanças com consultas de bancos de dados relacionais:

```
SELECT FROM Person WHERE name LIKE 'Luk%'
```

A consulta acima retorna todas as instâncias da classe Person onde o atributo name começa com a *string* 'Luk'. Finalmente, no que diz respeito a arquitetura distribuída, o OrientDB possui suporte em sua versão open source, enquanto o Neo4j possui suporte somente na licensa comercial. O OrientDB utiliza o método de *sharding* para separar os dados em clusters diferentes, enquanto o Neo4j não possui suporte para esse tipo de funcionalidade. Em relação aos métodos de replicação o OrientDB trabalha com o modelo multi-master, discutido em detalhes na seção 2.4.3, e o Neo4j utiliza o método *Causal Clustering using Raft protocol* [27]. Por possuir suporte a funcionalidade de *sharding*, o OrientDB também suporta operações de MapReduce sem o uso de tecnologias como o Hadoop ou Spark. Essa funcionalidade é totalmente transparente para o desenvolvedor, de forma que quando uma consulta envolve múltiplos clusters, o SGBD executa a query sobre cada nó (Operação de Map) e em seguida realiza o *merge* dos resultados (Operação de reduce) [28].

2.3.3 Conclusão

Portanto, em vista dos fatos mencionados, o OrientDB possui uma quantidade de funcionalidades maior que o Neo4j, principalmente em sua versão comunitária, ou open-source. Funcionalidades importantes como o uso de uma arquitetura distribuída, estão presentes somente na versão comercial do Neo4j, o que fez com que o OrientDB fosse escolhido para uso no estudo de caso. O fato de possuir uma linguagem próxima ao SQL, também pesou na escolha, pois proporciona uma produtividade maior para aqueles acostumados com a linguagem de consulta SQL.

Outra característica pouco pontuada, mas bastante importante, é a presença de uma comunidade por trás da tecnologia para auxiliar os desenvolvedores. A comunidade por trás do OrientDB é grande e bastante presente em sites como o *StackOverflow* [29] e *Linkedin*, inclusive com presença do próprio *CEO* da empresa em ambos os *sites*. Contudo, é inquestionável que o Neo4j é mais popular entre desenvolvedores e possui uma comunidade maior que a do OrientDB, o que sem dúvida é uma vantagem competitiva.

Uma conclusão em relação a qual tecnologia é melhor, seria que no geral ambas são igualmente boas com pontos positivos e negativos para cada lado. O fato do Neo4j possuir uma comunidade maior é bastante importante, mas o fato de poder utilizar um SGBD com arquitetura distribuída gratuitamente também tem sua importância. A escolha entre um ou outro deve ser ponderada de projeto a projeto, e para esse trabalho foi escolhido o OrientDB pelo motivos pontuados no início dessa seção.

2.4 OrientDB

Essa seção tem por objetivo especificar as características do SGBD OrientDB, e realizar uma comparação com outros SGBD orientado a grafos como o Neo4j.

2.4.1 Introdução

O OrientDB, é um SGBD de código aberto sob a licensa Apache. Ele é o primeiro SGBD NoSQL multi modelo, com suporte a uma arquitetura distribuída e orientado a grafos. Sendo assim o OrientDB suporta operações com documentos, chave/valor e grafos. Essa característica garante uma enorme flexibilidade para manipular os dados dentro do OrientDB, sendo possível armazenar os dados tanto como grafos ou como documentos no mesmo banco de dados [30]. A figura a seguir resume as possibilidades de registros no OrientDB:

O OrientDB é implementado utilizando a linguagem java, tendo sua primeira versão disponível no ano de 2010. Ele possui alta flexibilidade para definir o esquema do banco

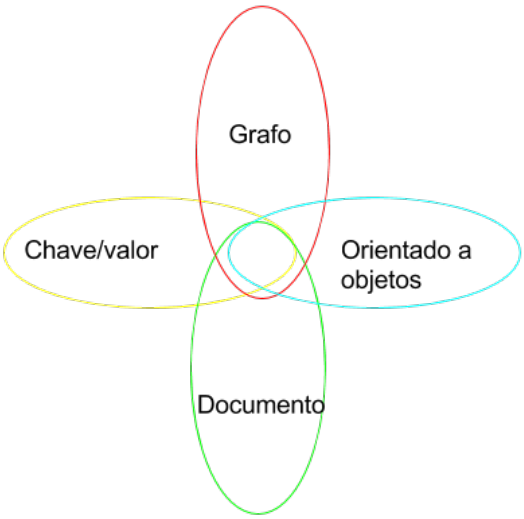

Figura 2.10: Possibilidade de registros no OrientDB

de dados, podendo ser *Schema-free*, *Schema-hybrid* ou *Schema-full*. A sua linguagem de consulta é derivada do SQL o que é bastante vantajoso para aqueles que possuem experiência com bancos de dados relacionais, e além disso ele utiliza o modelo de transações ACID que como foi mencionado na seção 2.2.3 é algo mais comum no grupo dos SGBD relacionais, isso demonstra que o OrientDB presa pela integridade dos dados ao mesmo tempo que também fornece um suporte a particionamento dos dados [30] [31].

Todas essas características fazem com que o OrientDB seja um SGBD bastante flexível e confiável para se utilizar em diversas aplicações. As operações utilizando grafos em específico, vem ganhando bastante visibilidade pois funciona muito bem em certos domínios de aplicação. Como foi mencionado no capítulo 1 e nesse artigo feito por Matthias Gelbmann [32] a popularidade dos SGBD orientados a grafos vem crescendo bastante nos últimos anos, e essas características ajudam a explicar o porque de banco de dados como o OrientDB e Neo4j estarem crescendo tanto em popularidade.

2.4.2 Performance

O OrientDB possui uma ótima performance em operações utilizando grafos. Um aspecto técnico que ajuda nessa qualidade é que o OrientDB trabalha cada registro como um objeto e o *link* entre esses objetos não é feito por referência, e sim por *link* direto. Ou seja, é salvo um ponteiro que aponta diretamente para o objeto referenciado. Isso faz com

que a velocidade para recuperar informações seja muito mais rápido em comparação com os joins utilizados nos SGBD relacionais. Portanto, não existem operações de joins dentro do OrientDB para obter as relações entre os registros, sendo que ele consegue salvar certa de 120 mil registro por segundo.

O OrientDB utiliza mecanismos de indexação baseados em árvores-B e hash extendido, esses mecanismos garantem uma complexidade constante para obter relacionamentos entre um registro para muitos registros. Um estudo feito pelo instituto de tecnologia de tokio e pela IBM, mostra que o OrientDB chega a ser 10 vezes mais rápido que o seu maior concorrente que é o Neo4j [33].

O trabalho feito po Toyotaro Suzumura e Miyuru Dayarathna [33], aponta como a estrutura de grafos vem crescendo em sistemas na nuvem, devido ao crescimento de aplicações que produzem dados em formato de grafos, como aplicações de web semântica, sistemas de informação geográfico (GIS), aplicação de bioinformática [34] e química informática [35]. Sendo que históricamente, dados com estrutura de grafos eram modelados com relacionamentos e armazenados em bases relacionais. Toda a lógica de percorrimento de grafos ficava para outras camadas da aplicação. Entretanto, percebe-se que o armazenamento desse tipo de dado e a análise feita em cima dessa estrurura feita nos SGBD são mais eficientes, uma vez que fornecem uma performance otimizada e produtividade na especificação de consultas [33].

Um dos objetivos desse artigo [33] é realizar um benchmark e estudar a performance de quatro grandes SGBD orientado a grafos em ambiente de nuvem. Os SGBD são AllegroGraph [36], Fuseki [37], Neo4j [38] e o OrientDB [39]. Eles utilizaram o XGDBench que é uma extensão do conhecido Yahoo! Cloud Serving Benchmark (YCSB). O YCSB é um framework de benchmark para sistemas na nuvem. Foram criados cinco *Workloads* para realizar os testes:

- *Update heavy*: 50/50 entre leitura e atualizações
- *Read mostly*: 95/5 entre leitura e atualizações
- *Read only*: 100 por cento de leituras
- *Read latest*: esse *Workload* insere novos vértices ao grafo
- *Short Ranges*: esse *Workload* lê todos os vizinhos e seus atributos de um dado vértice A.

Em todos os workloads, levando em conta a arquitetura e o hardware utilizado no ambiente de nuvem, o OrientDB teve um desempenho melhor que os demais SGBD [33].

2.4.3 Arquitetura Distribuída

Em relação ao suporte a uma arquitetura distribuída, o OrientDB trabalha com um método de particionamento conhecido como *sharding*, e um método de replicação conhecido como *Multi-master replication*. Esse modelo de replicação, permite que qualquer membro do cluster possa ler e escrever no banco de dados. Essa arquitetura permite que exista uma escalabilidade horizontal sem gargalos, como ocorre em algumas outras soluções de SGBD NoSQL [14].

No ano de 2012 o OrientDB trabalhava com uma arquitetura diferente, conhecida como *Master-slave replication*. A figura a seguir exemplifica o formato dessa arquitetura.

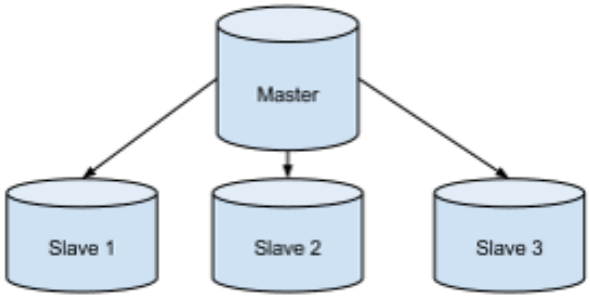

Figura 2.11: Exemplo de uma arquitetura *Master-slave*

Esse tipo de arquitetura é eficiente para escalar as operações de leituras, mas também é importante conseguir escalar as operações de escrita no banco de dados. Como é possível ver na imagem acima, um nó (*Master*) é responsável por receber as requisições de leitura e distribuir essa operações entre os demais nós *Slaves*. Porém, ao projetar dessa forma o nó *Master* se torna um gargalo para a aplicação, de forma que, ao receber muitas requisições o nó passa a ficar sobrecarregado.

Uma vantagem dessa arquitetura, é sua facilidade de implementação, pois basta realizar o roteamento das requisições entre os nós *Slaves*. Como desvantagens temos o fato do nó *Master* ser o gargalo nas operações de escrita, e a característica de que não adianta aumentar a quantidade de servidores para aumentar a vazão, pois a vazão é inevitavelmente limitada pelo nó *Master*.

Nesse cenário, o SGBD evoluiu para a arquitetura *Multi-master* representada pela figura acima. Nesse formato, todos os nós de um cluster aceitam operações de escrita. Além desse formato, o SGBD passou a adotar o método de particionamento conhecido como *sharding*, em que os dados são divididos em múltiplas partições. Como é mencionado na seção 2.4.4, o conceito de classes é bastante útil na arquitetura distribuída, uma vez que por padrão para cada classe criada o SGBD cria um cluster para que suas instâncias sejam armazenadas. Combinando essas técnicas, o OrientDB proporciona uma escalabilidade

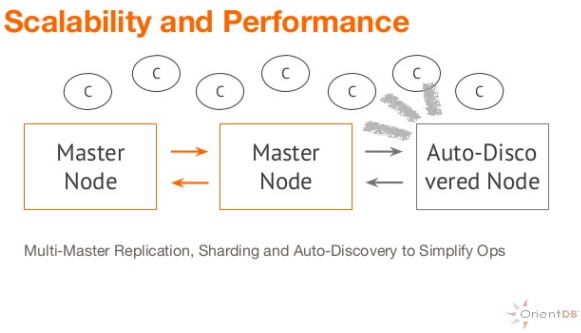

Figura 2.12: Exemplo de uma arquitetura *Multi-master*

tanto em leituras quanto em escritas. Uma vantagem desse modelo é que se um nó *Master* falhar, os demais podem continuar a operar normalmente.

2.4.4 Orientação a Objetos

Uma das características mais interessantes do OrientDB é o suporte que ele da para que o usuário organize os seus dados seguindo padrões de orientação a objetos. Como foi mencionado na seção 2.4.2, o OrientDB considera cada registro como um objeto, dessa forma é possível criar classes que representam a estrutura dos dados. Por exemplo, no capítulo 3 eu explico um pouco mais sobre como fiz a modelagem dos dados seguindo o modelo MDG-NoSQL proposto por Gustavo C. Galvão Van Erven[40]. Nessa modelagem eu identifiquei as seguintes classes Transação, Empresa fornecedora, Parlamentar e Pessoa. O OrientDB permite que eu crie essas classes e ao inserir um vértice eu especifique que esse vértice é de uma classe específica, isso facilita bastante na hora de escrever as consultas pois basta utilizar o nome da classe que todos os vértices pertencentes a essa classe serão obtidos. O conceito de classe também se aplica as arestas, e tudo isso facilita na organização dos dados e na escrita de consultas ao banco de dados.

Um registro é a menor unidade que é possível salvar e obter no banco de dados do OrientDB, ele pode ser obtido de quatro formas:

- Documento
- *RecordBytes* (BLOB)
- Vértice
- Aresta

Dessa forma, uma classe para esse tipo de registro, é o conceito mais próximo de tabela existente no OrientDB. O conceito de herança, muito utilizado no paradigma orientado a objetos, também possui suporte no OrientDB, sendo que classes podem herdar atributos

e propriedades de uma classe pai. Por exemplo, a classe pessoa mencionada acima, pode ser a classe pai da classe parlamentar, assim um parlamentar herda atributos de pessoa. Além disso, ao realizar uma consulta utilizando a classe pessoa, automaticamente todos os parlamentares também serão utilizados. Outra funcionalidade atrelada a esse assunto, é o suporte a classes abstratas utilizadas para definir outras classes. Uma classe abstrata não possui instâncias no banco de dados, bastante similar com os conceitos em orientação a objetos.

A figura a seguir exemplifica a herança presente no OrientDB, no caso é possível criar três classes para armazenar as instâncias dos vértices, que são: *Employee*, *Regular employee* e *Contract employee*. As duas últimas classes herdam os atributos id e name da classe pai, sendo que ao realizar uma busca por todos as instâncias de *Employee*, tanto *Regular employee* quanto *Contract employee* serão retornados.

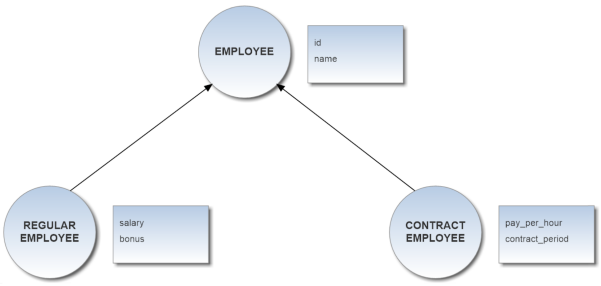

Figura 2.13: Exemplo de herança no *schema* do OrientDB

Para finalizar é importante mencionar que as classes tem um papel importante ao se utilizar a arquitetura distribuída no OrientDB, pois para cada classe criada, o SGBD cria um cluster automaticamente para armazenar instâncias dessas classes. Essa funcionalidade auxilía bastante na organização do particionamento dos dados em um ambiente distribuído. Todos os registros de uma classe são armazenados juntos no mesmo cliuster que tem o mesmo nome da classe. No OrientDB é possível criar até trinta e dois mil setecentos e setenta e sete clusters. Compreender bem os conceitos de classes nesse SGBD, permite que o usuário tenha vantagem ao fazer o *design* do banco de dados. Vale lembrar que uma classe pode ser mapeada para n clusters como mostra a figura a seguir:

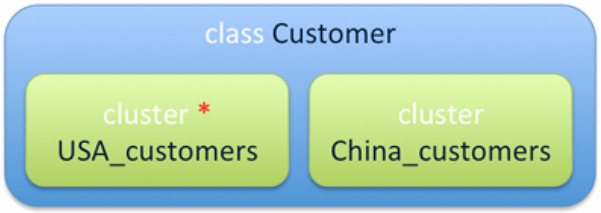

Figura 2.14: Classe Customer mapeada para dois clusters diferentes

2.5 *REST*

2.5.1 Introdução

O termo *REST* significa *Representational State Transfer*, em português Transferência de Estado Representacional, e foi introduzido por Roy Fielding em sua tese de doutorado [41]. Como Roy Fielding afirma em sua tese, a complexidade dos sistemas de *software* modernos fazem necessário uma ênfase maior em sistemas componentizados, em que a implementação é dividida em componentes independentes que se comunicam para realizar uma tarefa desejada [41].

Ainda em sua tese, Roy Fielding explora o encontro entre duas discplinas da área de ciência da computação: *software* e *networking*. Ele afirma que as pesquisas de softwares se preocupavam em categorizar o *desing* de *software*, e desenvolver novas metodologias de *desing*, em vez de avaliar o impacto das decisões de *design* no comportamento de um sistema. Enquanto as pesquisas de *networking*, foca nos detalhes da comunicação genérica entre sistemas e em melhorar a performance de uma técnica de comunicação específica, muitas vezes ignorando o fato de que mudar o estilo de interação de uma aplicação pode gerar mais impacto na performance, do que quais protocolos de comunicação são usados nessa interação [41].

Portanto, nesse contexto, o trabalho de Roy Fielding tinha como objetivo entender e avaliar o *design* arquitetural de uma aplicação de software baseada na rede através do uso de restrições arquiteturais, dessa forma obtendo as propriedades sociais, performáticas e funcionais que se espera de uma arquitetura [41]. Dessa forma, o *REST* é um estilo de arquitetura para sistemas hipermídia distribuídos. O *REST* provê um conjunto de restrições arquiteturais que ao serem aplicadas, enfatizam a escalabilidade das interações entre componentes, generalidade de interfaces, *deploy* independente dos componentes e componentes intermediários para reduzir a latência nas interações, reforçar a segurança e encapsular sistemas legados [41].

2.5.2 Princípios do estilo arquitetural *REST*

A característica principal que diferencia o *REST* de outros estilos baseados em rede, é a ênfase em uma interface uniforme entre os componentes. Dessa forma, a arquitetura geral do sistema é simplificada e a visibilidade das interações e melhorada. Entretanto, um problema nesse princípio é que a eficiência se torna pior. Isso ocorre, pois, a informação é transferida de forma padronizada, em vez de um formato específico a necessidade das aplicações [41].

Como foi dito na seção 2.5.1, o estilo arquitetural *REST* impõe uma série de restrições ao sistema. Portanto, o *REST* possui alguns princípios que os sistemas devem adotar para serem considerados *RESTful*, que são:

- Todos os componentes do sistema se comunicam através de interfaces com métodos bem definidos e código dinâmico. A comunicação entre os componentes utiliza o protocolo *HTTP*.
- Cada componente é unicamente identificado por um link hipermídia (URL)
- Uma arquitetura cliente/servidor *stateless* é seguida.
- A arquitetura possui camadas, e os dados podem ser cacheados em qualquer camada.

Atualmente, vários serviços na web fornecem interfaces *REST*, para que interessados possam consumir os dados, tais como a Amazon, eBay, Yahoo. A câmara dos deputados também utiliza o estilo arquitetural *REST* para fornecer algumas informações. Além disso, o framework *ruby on rails*, que foi utilizado para desenvolver o sistema do estudo de caso, suporta aplicações *REST* utilizando o padrão MVC.

Um ponto interessante a ser colocado, é que nem todas essas aplicações mencionadas são puramente *REST*, pois não respeitam todas as restrições destacadas por Roy Fielding em sua tese [41]. Esses serviços, seguem os princípios mais importantes, principalmente a interface uniforme. Tais aplicações são conhecidas como "acidentalmente *RESTful*"[42].

A figura abaixo exemplifica o estilo arquitetural *REST*:

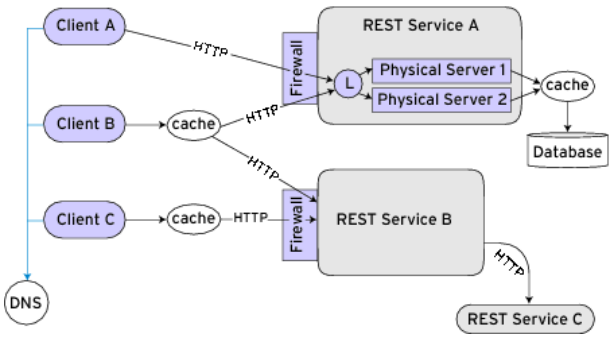

Figura 2.15: Estilo arquitetural *REST*

2.5.3 Interface *REST* no OrientDB

O SGBD orientado a grafos utilizado no estudo de caso foi o OrientDB, descrito em mais detalhes na seção 2.4. O OrientDB fornece uma interface *REST* de forma que as aplicações se comuniquem com o SGBD de forma eficiente e escalável. Essa interface

utiliza os métodos do protocolo *HTTP* para realizar operações nos dados, como descrito abaixo:

- Método GET: Utilizado para recuperar dados do banco de dados. É uma operação idempotente, isso significa que nenhuma mudança no banco de dados é feita.
- Método POST: Utilizado para persistir dados no banco de dados.
- Método PUT: Utilizado para atualizar dados já persistidos no banco de dados.
- Método DELETE: Utilizado para deletar dados já persistidos no banco de dados.

Dessa forma, uma aplicação que deseja se conectar ao SGBD utilizando a interface *REST*, realizaria o seguinte comando:

```
GET http://{{server}}:{{port}}/connect/{{database}}
```

Esse método se conecta com um servidor remoto usando o formato de autenticação básica. Basta fornecer o endereço e porta do servidor e o nome do banco de dados. Se tudo ocorrer corretamente uma resposta 204 OK é retornada.

Se a aplicação deseja, por exemplo, obter todos os parlamentares persistidos no banco de dados, realizaria o seguinte comando:

```
GET http://{{server}}:{{port}}/query/{{database}}/{{language}}/SELECT
from Parlamentar
```

Nesse caso, ao fornecer o endereço e porta do servidor, o nome do banco de dados e especificar a linguagem como sendo "sql", o SGBD retorna no formato JSON todos os parlamentares disponíveis no banco de dados.

Capítulo 3

CEAP: Detecção de fraudes com o uso de SGBD orientado a grafos

Capítulo 4

CEAP: Análise dos resultados obtidos

Capítulo 5

Conclusão

Referências

[1] Abramo, Cláudio Weber: *Relações entre índices de percepção de corrupção e outros indicadores em onze países da américa latina*. SPECK, Bruno W. et al. Os custos da corrupção. Cadernos Adenauer, (10):47–62, 2000. 1

[2] Filgueiras, Fernando: *A tolerância à corrupção no brasil: uma antinomia entre normas morais e prática social*. Opinião Pública, 15(2):386–421, 2009. 1

[3] engines db: *Dbms popularity broken down by database model*. `https://db-engines.com/en/ranking_categories`. Acessado em setembro de 2017. 2, 13

[4] Bondy, John Adrian, Uppaluri Siva Ramachandra Murty *et al.*: *Graph theory with applications*, volume 290. Citeseer, 1976. 4, 5, 6

[5] OrientDB: *Shotest path*. `https://orientdb.com/docs/3.0.x/gettingstarted/demodb/queries/DemoDB-Queries-Shortest-Paths.html`. Acessado em fevereiro de 2018. 7

[6] Strozzi, Carlo: *Nosql a relational database management system*. `http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql`. Acessado em janeiro de 2018. 7

[7] Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes e Robert E. Gruber: *Bigtable: A distributed storage system for structured data*. ACM Trans. Comput. Syst., 26(2):4:1–4:26, junho 2008, ISSN 0734-2071. `http://doi.acm.org/10.1145/1365815.1365816`. 7, 10

[8] DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall e Werner Vogels: *Dynamo: Amazon's highly available key-value store*. SIGOPS Oper. Syst. Rev., 41(6):205–220, outubro 2007, ISSN 0163-5980. `http://doi.acm.org/10.1145/1323293.1294281`. 7, 10

[9] Cattell, Rick: *Scalable sql and nosql data stores*. SIGMOD Rec., 39(4):12–27, maio 2011, ISSN 0163-5808. `http://doi.acm.org/10.1145/1978915.1978919`. 8

[10] Ramez Elmasri, Shamkant B Navathe: *Sistemas de banco de dados*. Pearson Addison Wesley, 2005. 8

[11] Pritchett, Dan: *Base: An acid alternative*. Queue, 6(3):48–55, maio 2008, ISSN 1542-7730. `http://doi.acm.org/10.1145/1394127.1394128`. 9

[12] Brewer, Eric: *A certain freedom: Thoughts on the cap theorem.* Em *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, páginas 335–335, New York, NY, USA, 2010. ACM, ISBN 978-1-60558-888-9. http://doi.acm.org/10.1145/1835698.1835701. 9

[13] Nayak, Ameya, Anil Poriya e Dikshay Poojary: *Type of nosql databases and its comparison with relational databases.* International Journal of Applied Information Systems, 5(4):16–19, 2013. 9, 10, 11, 12, 13

[14] Kaur, Amrinder e Rakesh Kumar: *Emerging no-sql technologies for big data processing.* 10, 20

[15] Kulshrestha, Sudhanshu e Shelly Sachdeva: *Performance comparison for data storage-db4o and mysql databases.* Em *Contemporary Computing (IC3), 2014 Seventh International Conference on*, páginas 166–170. IEEE, 2014. 13

[16] Edlich, Stefan, Reidar Hörning e Henrik Hörning: *The definitive guide to db4o.* Springer, 2006. 13

[17] engines db: *Db-engines ranking of graph dbms.* https://db-engines.com/en/ranking/graph+dbms. Acessado em fevereiro de 2018. 13

[18] Jouili, Salim e Valentin Vansteenberghe: *An empirical comparison of graph databases.* Em *Social Computing (SocialCom), 2013 International Conference on*, páginas 708–715. IEEE, 2013. 13

[19] Kolomičenko, Vojtěch, Martin Svoboda e Irena Holubová Mlỳnková: *Experimental comparison of graph databases.* Em *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, página 115. ACM, 2013. 13

[20] Kovacs, Kristof: *Cassandra vs mongodb vs couchdb vs redis vs riak vs hbase vs couchbase vs orientdb vs aerospike vs neo4j vs hypertable vs elasticsearch vs accumulo vs voltdb vs scalaris comparison.* Sitio web de Kristof Kovacs, 2016. 13

[21] Barmpis, Konstantinos e Dimitrios S Kolovos: *Evaluation of contemporary graph databases for efficient persistence of large-scale models.* Journal of Object Technology, 13(3):3–1, 2014. 13

[22] Labute, MX e MJ Dombroski: *Review of graph databases for big data dynamic entity scoring.* Relatório Técnico, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2014. 13

[23] Angles, Renzo: *A comparison of current graph database models.* Em *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, páginas 171–177. IEEE, 2012. 14

[24] Neo4j: *3.5. schema.* https://neo4j.com/docs/developer-manual/current/cypher/schema/. Acessado em fevereiro de 2018. 14

[25] OrientDB: *Sql - create property.* http://orientdb.com/docs/last/SQL-Create-Property.html. Acessado em fevereiro de 2018. 14

[26] OrientDB: *Live query.* http://orientdb.com/docs/last/Live-Query.html. Acessado em fevereiro de 2018. 14

[27] Neo4j: *Causal cluster lifecycle.* https://neo4j.com/docs/operations-manual/current/clustering/causal-clustering/lifecycle/. Acessado em fevereiro de 2018. 16

[28] OrientDB: *Sharding.* https://orientdb.com/docs/2.2/Distributed-Sharding.html. Acessado em fevereiro de 2018. 16

[29] *Orientdb tagged questions.* https://stackoverflow.com/questions/tagged/orientdb. Acessado em fevereiro de 2018. 17

[30] engines db: *Orientdb system properties.* https://db-engines.com/en/system/OrientDB. Acessado em janeiro de 2018. 17, 18

[31] chart vs: *Orientdb compare.* http://vschart.com/compare/orientdb. Acessado em janeiro de 2018. 18

[32] Gelbmann, Matthias: *Graph dbmss are gaining in popularity faster than any other database category.* https://db-engines.com/en/blog_post/26. Acessado em janeiro de 2018. 18

[33] Dayarathna, Miyuru e Toyotaro Suzumura: *Xgdbench: A benchmarking platform for graph stores in exascale clouds.* Em *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, páginas 363–370. IEEE, 2012. 19

[34] Dudley, Joel T, Yannick Pouliot, Rong Chen, Alexander A Morgan e Atul J Butte: *Translational bioinformatics in the cloud: an affordable alternative.* Genome medicine, 2(8):51, 2010. 19

[35] Ekins, Sean, Rishi R Gupta, Eric Gifford, Barry A Bunin e Chris L Waller: *Chemical space: missing pieces in cheminformatics.* Pharmaceutical research, 27(10):2035–2039, 2010. 19

[36] F. inc: *Fuseki: serving rdf data over http.* https://franz.com/agraph/allegrograph/. Acessado em janeiro de 2018. 19

[37] apache fuseki: *Allegrograph rdf store for web 3.0's database.* https://jena.apache.org/documentation/serving_data/. Acessado em janeiro de 2018. 19

[38] neo4j.org: *Neo4j: the world's leading graph database.* https://neo4j.com/. Acessado em janeiro de 2018. 19

[39] O. technologies: *Orientdb - the world's first distributed multi-model nosql database with a graph database engine.* http://orientdb.com/orientdb/. Acessado em janeiro de 2018. 19

[40] Erven, Gustavo C. Galvão van: *Mdg-nosql: modelo de dados para bancos nosql baseados em grafos*. Em *Dissertação (Mestrado Profissional em Computação Aplicada)—Universidade de Brasília*. Universidade de Brasília, 2015. 21

[41] Fielding, Roy T e Richard N Taylor: *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000. 23, 24

[42] Baker, Mark: *Accidentally restful*. `http://www.markbaker.ca/blog/2005/04/accidentally-restful/`. Acessado em fevereiro de 2018. 24