CLÁUDIA MORGADO

MODELO DE SEGURANÇA PARA BANCO DE DADOS ORIENTADO A GRAFOS

LIMEIRA

2016

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA
MESTRADO EM TECNOLOGIA

CLÁUDIA MORGADO

Modelo de Segurança para Banco de Dados Orientado a Grafos

Dissertação apresentada ao Curso de Mestrado da Faculdade de Tecnologia da Universidade Estadual de Campinas, como requisito parcial para a obtenção do título de Mestra em Tecnologia.

Área de Concentração: Tecnologia e Inovação

Supervisora/orientadora: Profª. Drª. Regina Lúcia de Oliveira Moraes

Co-supervisora/coorientadora: Prof.ª Dr.ª Gisele Busichia Baioco

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DISSERTAÇÃO DEFENDIDA PELA ALUNA CLÁUDIA MORGADO, ORIENTADA PELO PROFª. DRª. REGINA LÚCIA DE OLIVEIRA MORAES E CORIENTADA PELA PROF.ª DR.ª GISELE BUSICHIA BAIOCO

LIMEIRA
2016

Ficha catalográfica

Universidade Estadual de Campinas

Biblioteca da Faculdade de Tecnologia

Silvana Moreira da Silva Soares - CRB 8/3965

M82

Morgado, Cláudia, 1977-

Modelo de segurança para banco de dados orientado a grafos / Cláudia Morgado. – Limeira, SP : [s.n.], 2016.

Orientador: Regina Lúcia de Oliveira Moraes.

Coorientador: Gisele Busichia Baioco.

Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Banco de dados. 2. Segurança. 3. Banco de dados - Medidas de segurança. I. Moraes, Regina Lúcia de Oliveira,1956-. II. Baioco, Gisele Busichia,1970-. III. Universidade Estadual de Campinas. Faculdade de Tecnologia. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Security model for graph-oriented databases
Palavras-chave em inglês:
Database
Security
Database - Security measures
Área de concentração: Sistemas de Informação e Comunicação
Titulação: Mestra em Tecnologia
Banca examinadora:
Gisele Busichia Baioco [Coorientador]
Luiz Camolesi Júnior
Alexandre Ferreira Mello
Data de defesa: 29-02-2016
Programa de Pós-Graduação: Tecnologia

DISSERTAÇÃO DO MESTRADO EM TECNOLOGIA

ÁREA DE CONCENTRAÇÃO: TECNOLOGIA E INOVAÇÃO

Modelo de Segurança para Banco de Dados Orientado a Grafos

Cláudia Morgado

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof.ª Dr.ª Gisele Busichia Baioco UNICAMP
Presidente

Prof. Dr. Luiz Camolesi Jr.
UNICAMP

Prof. Dr. Alexandre Mello Ferreira
EEP

A ata da defesa com as respectivas assinaturas dos membros encontra-se no processo de vida acadêmica da aluna.

Agradecimentos

Primeiramente a Deus, por me dar força permitindo que eu realizasse mais uma conquista, e colocar anjos no meu caminho que me ajudaram no decorrer de todo este trabalho.

A minha orientadora Profª. Drª. Regina Lúcia de Oliveira Moraes e coorientadora Prof.ª Dr.ª Gisele Busichia Baioco que me orientaram sem medir esforços durante toda a minha caminhada.

A universidade UNICAMP que me proporcionou um mestrado de qualidade.

Ao meu amigo Lucas Rodrigues Paiva que me ajudou na parte prática da implementação do modelo de controle de acesso.

A minha família que me compreendeu a disposição de tempo que dediquei a este trabalho, em especial a minha irmã Silvana Morgado dos Santos e a meu marido Evandro Roberto da Silva pela paciência e dedicação.

Aos meus pais, minha base, que sempre acreditaram e apoiaram todas as etapas da minha vida.

Que Deus abençoe a todos.

Resumo

A cada ano mais informações digitais são criadas, informações essas que já não preservam a estrutura fixa e bem definida de antes. O mundo nunca lidou com tanto volume de dados, semi-estruturados ou não estruturados. Os gerenciadores de banco de dados relacionais já não são mais eficientes para essas aplicações de requisitos complexos que requerem grande escalabilidade e disponibilidade.

Surge então um novo paradigma de banco de dados, os não somente relacionais denominados NoSQL, mais simples e flexíveis que os relacionais. Vários requisitos nativos a gerenciadores de banco de dados relacionais como, o controle de acesso, são deixados, nos gerenciadores não relacionais, para serem desenvolvidos pela aplicação, de acordo com necessidades específicas.

O controle de acesso aos dados é considerado um requisito essencial para a segurança da informação. Para garantir um mecanismo que iniba usuários não autorizados de acessar e modificar as informações de dados restritos, neste trabalho é definido com auxilio de metadados um modelo de controle de acesso para um sistema de gerenciamento de banco de dados NoSQL orientado a grafos (SGBDG).

Dessa forma, além de proporcionar maior agilidade e facilidade no desenvolvimento de aplicações que utilizem o SGBDG, a solução auxilia na segurança e privacidade ajudando a preservar a integridade dos dados armazenados.

Palavras-chave: Segurança, NoSQL, controle de acesso, Neo4j.

Abstract

Nowadays digital information has being created and this information did not preserve the structure stable and well defined as used before. The world had never dealt with so much data, semi-structured or unstructured and the relational database management systems are no longer efficient for these applications that are dealing with complex requirements and demand high scalability and availability.

Then comes up a new paradigm of database management systems, called the not only relational NoSQL, simpler and more flexible than the relational one. Several native requirements in relational database, for example, access control, has been left in nonrelational management systems, to be developed by the application according to their needs.

The access control is a key requirement for information security. To ensure a mechanism that inhibits unauthorized users to access and modify the information and restricted data, this paper proposes the use of metadata to create a access control model for a NoSQL, a graph-oriented databases management system (SGBDG) .

Thus, in addition to providing greater agility and ease the development of applications that use the NoSQL SGBDG, the proposed solution will aid in preserving the integrity of stored data.

Key words: Security, NoSQL, Access Control, Neo4j.

Índice

Índice de Figuras

Índice de Tabela

1. Introdução

O sistema de gerenciamento de banco de dados (SGBD) relacional foi criado na década de 1970, quando as aplicações de banco de dados caracterizavam-se por lidar com dados estruturados, ou seja, que possuem uma estrutura fixa e bem definida (Lóscio et al, 2011). Desde então, sua popularidade disparou e o SGBD se tornou a estrutura de base de dados mais utilizada no mundo acadêmico e comercial até os dias de hoje (Vicknair et al, 2010).

Com a evolução potencial do modelo da computação tradicional, em parte graças a WEB 2.0, houve uma mudança na maneira de utilizar os recursos da WEB. As redes sociais, por exemplo, requerem gerenciamento de dados não estruturados, os quais são gerados, diariamente, por milhões de usuários. O número de sistemas que são formados por entidades automaticamente relacionadas e que devem ser persistidas e consultadas aumentou significativamente.

A estrutura de dados relacional passou a ser ineficiente quando a base de dados contém muitos relacionamentos entre tabelas com grandes volumes de dados, ou quando há necessidade de armazenamento de dados que podem ser naturalmente representadas em uma estrutura de grafos (Vicknair et al, 2010; Leavitt, 2010).

Requisitos tidos como indiscutíveis foram revistos e surgiram então novos conceitos de gerenciadores de banco de dados não somente relacionais (NoSQL) e diferentes formas para persistir os dados.

O termo NoSQL passou a ganhar popularidade no inicio de 2009 e vem conquistando a comunidade de TI (Tecnologia da Informação) como uma nova forma de armazenamento de dados. Embora ainda não tenha sido alvo de grande número de publicações no mundo acadêmico, conta com seus próprios grupos de discussões, *blogs* e conferências (Vicknair et al, 2010).

Segundo Vicknair (2010), é aconselhável utilizar o gerenciador de banco de dados não somente relacional quando a base de dados possui as seguintes características:

- Tabelas com muitas colunas, cada uma delas usadas em poucos registros,

- Muitos relacionamentos em tabelas com grande volume de dados,
- Grande quantidade de tabelas com relacionamento muitos-para-muitos,
- O modelo de dados apresentando características em árvore,
- Grande necessidade de mudança no modelo de dados.

Uma das maiores dificuldades dos SGBDs NoSQL, nos dias de hoje, é a segurança. Segundo Zahid, et al (2014), não existe uma solução completa para os SGDB NoSQL, alguns não possuem nenhum sistema de segurança implementado e outros que possuem, ainda estão em um processo evolutivo. Entretanto, tem se tornado responsabilidade dos clientes dos SGBDs NoSQL proteger a base de dados usando ferramentas e serviços de terceiros. O aumento da demanda na utilização dos SGBDs NoSQL orientado a grafos (SGBDG) e a importância da segurança motiva a pesquisa com esse foco.

Ocorrências recentes mostraram que uma falha em um nó do grafo pode causar ou aumentar a probabilidade de ocorrência de defeitos em outros nós. Podemos citar vários exemplos de perdas significativas em sistemas com características de redes complexas, adequadas para serem representadas em grafos, que sofreram grande impacto em virtude de falhas que ao serem ativadas geraram erros que se espalharam pela rede resultando em defeito no sistema. Um exemplo seria o apagão tecnológico ocorrido em três de julho de 2008, quando o estado mais populoso do Brasil, com mais de 40 milhões de habitantes ficou sem Internet. Outro exemplo, seria a total paralisação do *Speedy*, quando serviços essenciais como bancos, agências da Previdência de 407 municípios e demais usuários do Speedy ficaram totalmente paralisados. A causa de tudo isso foram falhas em seis roteadores que foram ativadas e geraram erros que se espalharam através da rede (Pereira, 2010).

Defeitos como estes podem acontecer porque a exclusão ou corrupção de determinados nós, podem isolar partes de nossa base de dados e torná-la inconsistente. Por exemplo, na Figura 1.1, se os nós assinalados em vermelho forem removidos, parte dos dados se tornará inacessível. Particularmente, se o nó central for removido, toda a base de dados estará corrompida e o estado

inconsistente gerado pela remoção do nó será propagado instantaneamente para toda a base de dados. Dessa forma, assim como nos gerenciadores de base de dados relacionais, o acesso aos dados e a permissão para efetuar determinadas operações precisam ser cuidadosamente tratados, de forma que pessoas não autorizadas ou não capacitadas tenham suas ações restringidas às necessidades que lhes competem.

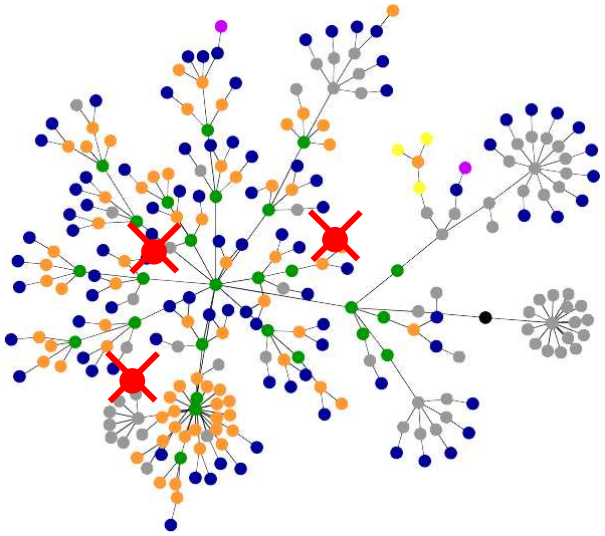

Figura 1.1: Representação de Dados em Estrutura de Grafos

A exposição dos dados armazenados em um banco de dados que não tenha implementada uma camada de segurança, representa uma vulnerabilidade substancial, uma vez que, a restrição necessária ao acesso e manipulação dos dados é delegada à aplicação. Essa delegação de responsabilidade ao desenvolvedor de sistemas foge do padrão que vinha sendo utilizado, até então, quando os gerenciadores de banco de dados relacionais, uma vez parametrizados, cuidavam automaticamente da segurança dos dados armazenados. Além disso, devido a ubiquidade dos sistemas atuais, as informações armazenadas seriam facilmente atacadas, caso um banco de dados, desprovido de mecanismos de segurança, fosse utilizado para sistemas reais.

1.1. Motivação

Os SGDBs NoSQL não vieram substituir os SGDBs Relacionais, mas sim, ser uma alternativa para aplicações com grande volume de dados semi-estruturados ou não estruturados.

Empresas estão adotando SGBDs NoSQL para atender sua demanda de escalabilidade, alta disponibilidade e armazenamento de dados não estruturados.

Existem diferentes tipos de SGBDs NoSQL e as características do modelo de dados fazem que um SGBD seja melhor para determinadas aplicações que outros. Por exemplo, em aplicações de dados estatísticos que são frequentemente escritos, mas raramente lidos, podem ser usados os SGBDs NoSQL orientado a chave/valor, ou um SGBD NoSQL orientado a documento. Em aplicações que exigem alto desempenho em consultas com muitas junções, pode-se adotar o SGDBG (Lóscio et al, 2011).

Existem diversas aplicações que podem se beneficiar do modelo de dados orientado a grafo, Newman (2003), dividiu-as em quatro classes distintas, sendo elas:

- **Redes Sociais:** neste tipo de rede os nós são compostos por pessoas ou grupos relacionados entre si. Alguns exemplos são as redes de amizades como Facebook, ou rede de negócios como Linkedin;

- **Redes de Informação:** as redes deste grupo modelam fluxos de informações. Elas são utilizadas, por exemplo, para relacionar citações de trabalhos acadêmicos e para relacionar classes de palavras em dicionários de sinônimos;

- **Redes Tecnológicas:** estas redes são caracterizadas pela importância dos aspectos geográficos e espaciais das estruturas. Alguns exemplos são: redes de computadores, redes de energia elétrica, rotas aéreas, redes de telefonia, sistemas de informação geográfico (GIS).

- **Redes Biológicas:** as redes biológicas representam informações biológicas onde existe um grande volume de dados de difícil gestão e análise. Estas redes ocorrem na regulação de genes, mapas metabólicos, estruturas químicas e relacionamento entre as espécies.

Alguns casos de sucesso podem ser vistos com a utilização do NoSQL, tais como o *Facebook* e *Twitter* que utilizam o Cassandra; mais de 60 produtos da Google (*Gmail, Google*

Docs, Google Earth) entre outros, utilizam o Big Table desenvolvido pela própria Google (Leavitt, 2010; Lóscio et al, 2011).

A segurança da informação é um dos temas mais discutidos nas organizações. Expor os dados armazenados em um gerenciador de banco de dados que não tenha implementado mecanismos de segurança, representa uma vulnerabilidade que pode inviabilizar o seu uso em determinados contextos.

A segurança (ou falta dela) em gerenciadores de banco de dados NoSQL é preocupante, pois todo o processo de acesso seguro e preservação do conteúdo é deixado para aplicação. A importância da segurança nos SGDBGs, bem como o aumento da demanda por esse modelo de gerenciador, motiva a pesquisa com esse foco.

1.2. Objetivo

Para contribuir com a melhoria do cenário retratado pela falta de segurança no controle de acesso ao banco de dados NoSQL, como exemplo o Neo4j, implementamos uma camada de segurança baseada em mecanismos de controle de autorização de acesso a dados. A importância desse tema pode ser entendida se acompanharmos os esforços demandados para a melhoria dos mecanismos similares implementados nos diferentes SGBDs relacionais, tais como Oracle, SQL Server e MySQL, por exemplo (Yang, 2009).

Este trabalho teve como objetivo a pesquisa e implementação das características essenciais de uma camada de controle de acesso para o SGBD NoSQL, provendo uma arquitetura que inclui regras de autorização para o controle de acesso à base de dados.

A idéia foi criar um modelo de controle de acesso, baseado em metadados com as regras de autorização de acesso aos dados, isto é, regras para diferentes perfis de acesso, tais como, os usuários, administradores, entre outros. Implementar este modelo de controle de acesso, como base de testes, no SGDBG Neo4j.

Como trabalho futuro espera que seja incrementado ao modelo de controle de acesso, metadados com regras para criação de esquema de relacionamentos, fazendo com que o controle de acesso, tenha restrições para diferentes perfis na manipulação de relacionamentos, assim como estamos fazendo com os nós. Como os usuários e senhas ficam armazenados na base de dados orientada a grafos (BDG), criptografar essas informações aumentaria a segurança.

1.3. Contribuições do Trabalho

As ameaças aos bancos de dados resultam na perda ou na degradação de alguns ou de todos os objetivos de segurança: integridade, disponibilidade e confiabilidade (ELMASRI e NAVATHE, 2005), onde:

- Perda de integridade: refere-se à modificação do banco de dados de maneira imprópria. A modificação de dados inclui a criação, a inclusão, a alteração, a mudança de status do dado e a exclusão;
- Perda de disponibilidade: refere-se em tornar os objetos indisponíveis para um usuário humano ou para um programa que tenham direito legítimos a eles;
- Perda da confiabilidade: refere-se à falta de proteção dos dados contra o uso não autorizado.

Para proteger o banco de dados contra esses tipos de ameaças, quatro tipos de medidas podem ser implementadas: controle de acesso, controle de inferência, controle de fluxo e criptografia (ELMASRI e NAVATHE, 2005). Neste trabalho o foco está no controle de acesso.

O resultado deste trabalho criou uma arquitetura complementar para SGDBG NoSQL, que é responsável pela autorização de acesso aos dados armazenados. O modelo de controle de acesso criado é independente e proporcionam maior agilidade e facilidade no desenvolvimento de aplicações que utilizam o SGBD NoSQL.

1.4. Organização do Trabalho

A presente dissertação de mestrado está organizada da seguinte forma:

Capítulo 2 – Revisão Bibliográfica. Explica, resumidamente, conceitos de grafos, banco de dados NoSQL, segurança em banco de dados, uma das bases teóricas desta dissertação.

Capítulo 3 – Trabalhos relacionados. Mostra os trabalhos que serviram de base para a presente dissertação.

Capítulo 4 – Modelo de segurança. Apresenta a modelo de controle de acesso.

Capítulo 5 – Estudo de Caso. Apresenta, a aplicação e validação do modelo de controle de acesso no SGBDG NoSQL.

Capítulo 6 – Conclusão e trabalhos futuros.

2. Revisão Bibliográfica

Para um melhor entendimento dessa pesquisa esta seção apresenta conceitos importantes para realização desse trabalho. Os temas abordados tem início na seção 2.1 com a definição de grafos. A seção 2.2 apresenta o modelo de dados para representação de grafos. Em seguida a seção 2.3 mostra a evolução dos bancos de dados NoSQL. A seção 2.4 apresenta os diferentes tipos de SGBDs NoSQL, com exemplos de SGBD para cada tipo. A sessão 2.5 descreve o SGBDG Neo4j, SGBDG utilizado como estudo de caso na implementação do modelo de controle de acesso apresentado nesta dissertação.

2.1. Definições e propriedades dos Grafos

A origem da teoria de grafos geralmente é associada ao problema das pontes de *Königsberg*, na Prussia. Parte desta cidade se localiza em duas ilhas do rio Pregel, que estão ligadas por 7 pontes. O problema residia em ser capaz de fazer um circuito completo pelo complexo de ilhas ligadas por estas sete pontes, de forma que não se repetisse a travessia de nenhuma ponte. A Figura 2.2 mostra que para elaborar o grafo, o matemático Leonhard Euler substituiu as partes de terra pôr vértices (nós) e as pontes pelas arestas que ligam os nós.

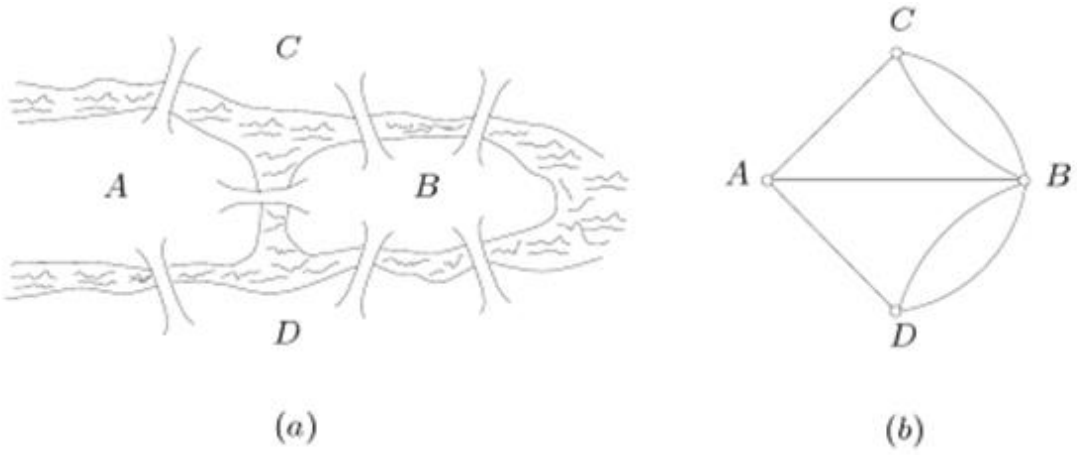

Figura 2.1: Problema das pontes de Königsberg tratado por Leonhard Euler (Boundy e Murty, 2008).

Se cada ponte for atravessada exatamente uma vez, cada massa de terra (nó) deveria ter um número par de arestas (relacionamento) conectadas a ele, com exceção apenas do nó de partida e chegada. Com isso o matemático suíço Leonhard Euler demonstrou, por meio da teoria dos grafos, ser impossível fazer esse percurso passando apenas uma vez pelas pontes (Bollobás, 1998; Newman, 2003; Boundy e Murty, 2008).

Muitas situações do mundo real podem ser representadas por diagramas formados por pontos (vértices) e arestas, unindo alguns pares desses pontos. Por exemplo, os pontos podem representar pessoas e a linha a comunicação entre elas (Boundy e Murty, 2008); se duas pessoas são unidas por uma linha, nesse contexto, significa que existe um canal de comunicação entre elas. Um grafo simples (G) e determinado por G = (V,E), onde o elemento V representa um conjunto finito e não vazio de vértices (nós) e E o conjunto de arestas (Bollobás, 1998 e Distel, 2005). Dois nós são **adjacentes**, se estão conectados por uma aresta (Wilson, 1996).

A Figura 2.1 representa um grafo onde os pontos são os vértices V= {1,.....,7} e o relacionamento entre eles as arestas E = {{1,2}, {1,5}, {2,5}, {3,4},{5,7}}.

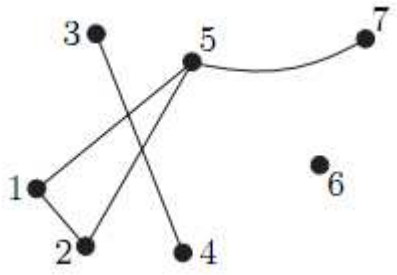

Figura 2.2: Representação do Grafo (Distel, 2005).

O **grau** de um vértice (nó) é um número de arestas incidentes nele. Um vértice isolado tem o grau 0. Na Figura 2.2 o vértice cinco V(5) possui grau três (3) por ter 3 arestas incidentes e o vértice seis v(6) possui grau 0 por não possuir nenhuma aresta incidente (Wilson, 1996). Os grafos podem ser direcionados ou não. A Figura 2.3 mostra um gráfico direcionado, em um grafo direcionado as arestas possuem setas que representam as direção da incidência no vértice. Por exemplo na Figura 2.3 dizemos que

2.2. Modelos de representação de grafos

O Modelo para representação de grafos são aplicados em áreas em que a interconectividade dos dados ou a topologia são mais importantes ou tão importantes quanto o dado em si (ANGLES e GUTIERREZ, 2008). Os modelos de dados devem representar as variações da matemática básica na definição de um grafo, como por exemplo, direcionamento ou não direcionamento do grafo, rótulos, relacionamentos, nós, hiper-grafos e hiper-nós.

O modelo de dados escolhido para representar o modelo de controle acesso deste trabalho foi o *Resource Description Framework* (RDF) por ser recomendado pelo W3C, e originalmente designado para representar metadados (ANGLES e GUTIERREZ, 2008).

O modelo RDF possui informações básicas da teoria dos grafos como nó, relacionamentos, caminhos, vizinhança e conectividade e são representados pelos símbolos como mostra a Figura 2.3, onde os atributos são representados por retângulos, os nós em si são representados por retângulos com os cantos arredondados e o relacionamento entre os nós uma seta indicando a direção do relacionamento.

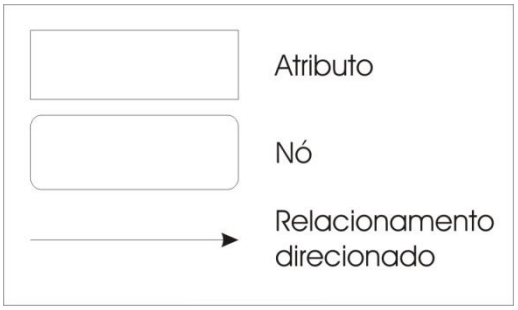

Figura 2.3: Símbolos do modelo de dados RDF

A Figura 2.4, mostra um exemplo do modelo RDF, onde o nó “Pessoa” possui dois atributos “Nome” e “Sobrenome” e ambos os atributos são do tipo String. O relacionamento entre esses nós acontece pelo parentesco entre os nós “Pessoa”. No exemplo instanciado, podemos ver a relação de parentesco entre Ana Silva, João Santos e Julia Santos.

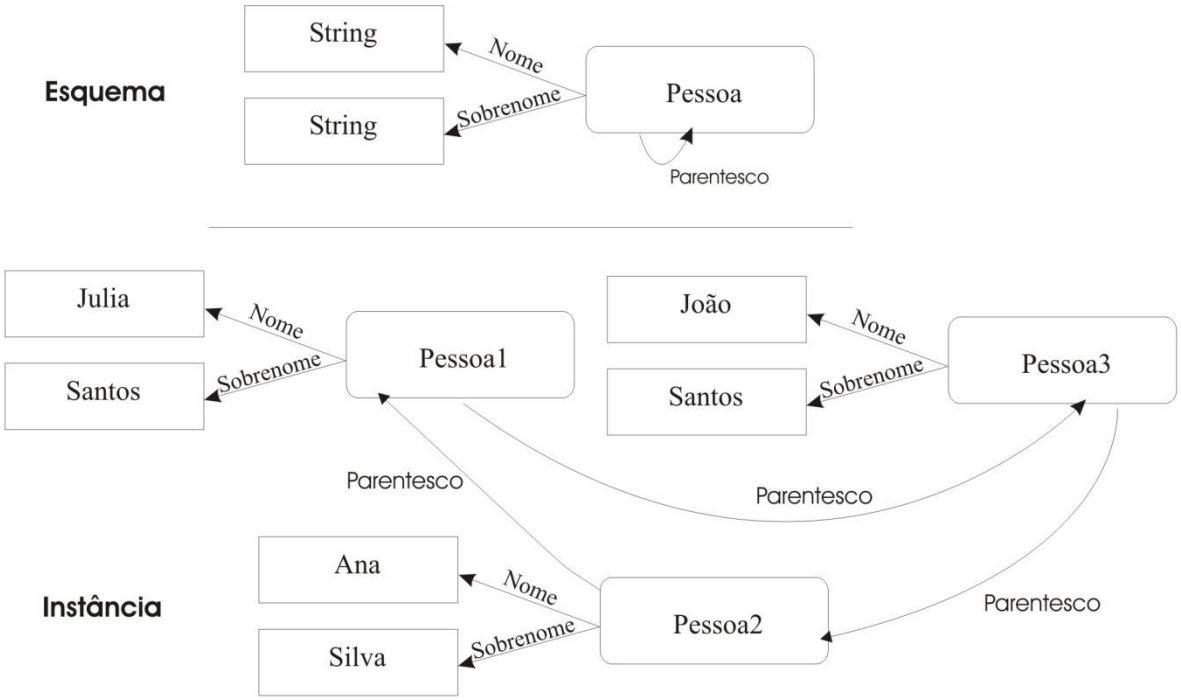

Figura 2.4: Exemplo de um modelo de dados RDF

2.3. NoSQL

Os SGBDs relacionais possuem limitações no gerenciamento de grande volume de dados, semi-estruturados ou não estruturados, que necessitam de alta disponibilidade e escalabilidade (Leavit, 2010).

O termo NoSQL define uma classe de banco de dados não somente relacional (do inglês *Not only SQL*) que surgiu em 2009 e tem sido reconhecido na área de banco de dados como uma alternativa aos SGBDs relacionais para o armazenamento de dados não estruturados, com grande volume de dados e necessidade de escalabilidade entre servidores de banco de dados. (Stonebraker, 2010; Lóscio et al, 2011).

O objetivo da solução NoSQL é prover uma forma eficiente de acesso aos dados, oferecendo alta disponibilidade e escalabilidade, ou seja, o foco não está em como os dados são armazenados e sim em como podemos recuperá-los de forma eficiente. Para isso conta com API simples para

acesso aos dados, permitindo que qualquer aplicação utilize os dados do banco de dados de forma rápida e eficiente (Lóscio et al, 2011).

2.3.1. NoSQL – conceitos e propriedades

Acredita-se que a base de dados NoSQL não substituirá a relacional, mas se tornará uma boa opção para certos tipos de projetos (Leavitt, 2010). Os bancos de dados NoSQL apresentam algumas características fundamentais que os diferenciam dos tradicionais sistemas de bancos de dados relacionais, tornando-os adequados para armazenamento de grandes volumes de dados não estruturados ou semiestruturados (Lóscio et al, 2011).

A medida que o volume de dados cresce, aumenta a necessidade de escalabilidade e melhoria de desempenho. Para a solução deste problema, temos a **escalabilidade vertical**, que consiste em aumentar o poder de processamento e armazenamento das máquinas e a **escalabilidade horizontal**, onde ocorre um aumento no número de máquinas disponíveis para o armazenamento e processamento de dados (Lóscio et al, 2011).

Outra característica evidente dos bancos de dados NoSQL é a ausência completa ou quase total do esquema que define a estrutura dos dados modelados. Esta ausência de esquema facilita tanto a escalabilidade quanto contribui para um maior aumento da disponibilidade. Em contrapartida, não há garantias da integridade dos dados, o que ocorre nos bancos relacionais, devido à sua estrutura rígida (Lóscio et al, 2011). Outra forma de prover a escalabilidade é através da replicação.

O objetivo da solução NoSQL é prover uma forma eficiente de acesso aos dados, oferecendo alta disponibilidade e escalabilidade, para isso conta com API simples de acesso aos dados. Os benefícios dos SGBDs NoSQL não vem sem custo, comparado com os bancos de dados tradicionais vamos perder alguma funcionalidade/garantia para ganhar outras.

O professor Dr. Eric Brewer, introduziu o Teorema de CAP, que explica que em qualquer sistema distribuído é preciso escolher entre consistência, alta disponibilidade e tolerância de particionamento de dados na rede. Segundo o teorema de CAP, entre essas três propriedades,

somente duas podem ser garantidas (Han et al, 2011 e Lóscio et al, 2011). Poder particionar nossos dados em diferentes nós de um *cluster* é um dos recursos que aparecem com frequência nos SGBDGs NoSQL. Saber lidar com o particionamento dos dados devido a uma falha na rede é conhecido como *Partition-Tolerant.* No entanto, segundo o teorema CAP, em troca eles irão sacrificar a consistência forte ou a alta disponibilidade. Isso é diferente dos bancos tradicionais, que não possuem essa característica no design do sistema.

Na Figura 2.5, podemos ver que sistemas que precisam da consistência forte e tolerância a particionamento **(CP)** não possuem o forte na alta disponibilidade, pode acontecer, caso haja particionamento e o sistema não entre em consenso, que uma escrita seja rejeitada. Exemplos desses sistemas CP são BigTable, HBaseou, MongoDB entre vários outros.

Por outro lado existem sistemas que jamais podem ficar *offline* (24/7), portanto não desejam sacrificar a disponibilidade. Para ter alta disponibilidade mesmo com uma tolerância a particionamento **(PA)**, é preciso prejudicar a consistência *(eventual-consistency).* A idéia aqui é que os sistemas aceitam escritas sempre e tentam sincronizar os dados em algum momento depois (read-consistency). Então pode ter uma janela de inconsistência. Exemplos aqui são Amazon Dynamo, Cassandra ou Riak.

Os sistemas com consistência forte e alta disponibilidade **(CA)** (alta disponibilidade de um nó apenas) não sabem lidar com a possível falha de uma partição. Caso ocorra, sistema inteiro pode ficar indisponível até o membro do *cluster* voltar. Exemplos disso são algumas configurações clássicas de bancos relacionais.

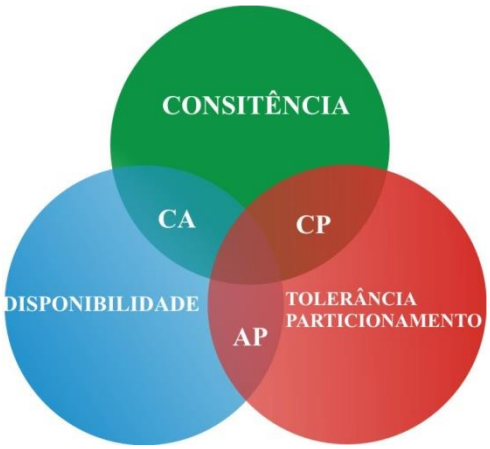

Figura 2.5: Teorema de CAP

A execução de transações em um SGBD relacional deve obedecer a algumas propriedades a fim de garantir o correto funcionamento do sistema e a respectiva consistência dos dados. Estas propriedades são chamadas de propriedade ACID e são definidas a seguir:

- **A**tomicidade: ou toda transação é feita ou nada é feito.
- **C**onsistência: as regras de banco de dados são asseguradas.
- **I**solamento: cada transação é executada de forma isolada de outra transação.
- **D**urabilidade: os efeitos de uma transação em caso de sucesso devem ser persistidos no banco de dados.

Estas propriedades são garantidas por alguns SGBDGs, como por exemplo, o SGBDG Neo4j.

2.3.2. SGBDs NoSQL

O mercado já conta atualmente com um número significativo de SGBDs NoSQL. Podemos enquadrar os bancos de dados NoSQL de acordo com os diferentes tipos de modelo de

dados. Existem quatro categorias de SGBDs NoSQL, modelo de dados orientado a chave-valor, coluna, documentos e grafos, as próximas seções explicam as características de cada modelo.

2.3.3. Modelo de dados Chave-Valor

O modelo chave-valor é um modelo simples, fácil de implementar e permite a visualização do banco de dados como uma tabela *hash* na qual há uma chave única e um indicador de um dado ou de um item em particular (Cattell, 2010; Lóscio,2011).

Alguns bancos que utilizam esse padrão são: DynamoDb, Membase, Riak, Azure Table Storage, Redis, Tokyo Cabinet, Scalaris, entre outros.

Um dos grandes desafios enfrentados pela Amazon, diz respeito à confiabilidade do grande volume de dados gerenciado por suas aplicações. Com intuito de garantir a disponibilidade dos dados de seus serviços, desenvolveu uma solução NoSQL, o Dynamo e após a adoção desta nova tecnologia seus dados têm se mantido disponíveis em 99,9995% das requisições realizadas (Loscio, 2011).

O Amazon DynamoDB é um serviço de banco de dados NoSQL de gestão completa que fornece um desempenho rápido e previsível com facilidade de escalabilidade. A Amazon disponibiliza vários serviços para gerenciamento do banco de dados, o que o torna simples de se utilizar, com custo compatível com o uso, uma vez que o cliente paga apenas por aquilo que usar (Dynamo, 2013).

O banco de dados Windows Azure Table Storage possui armazenamento persistente por meio de *blobs*, tabelas e filas (Sousa et al, 2011). Um *blob* é um par “nome, objeto”, que permite armazenar objetos de até 50 GB (Sousa et al, 2011). As tabelas são diferentes do relacional, armazenam um grande volume de dados que não necessitam de associações complexas ou chave estrangeira (Sousa et al, 2011 e Azure, 2013).

O Redis e o Tokyo Cabinet são banco de dados *open-source* que também utilizam o modelo de dados chave-valor. Possuem uma base de dados simples com registros, cada um representado por um par de chave e valor (Redis, 2013 e Tokyo, 2013).

2.3.4. Modelo de Dados Orientado a Colunas

O modelo de dados orientado a colunas é indexado por uma linha com atributos (colunas) e a data/hora de inclusão dos dados para diferenciar as múltiplas versões dos dados. As operações de leitura e escrita são atômicas, ou seja, todos os valores associados a uma linha são considerados na execução dessas operações.

Alguns bancos de dados que utilizam essa tecnologia são: Cassandra, BigTables da Google, Apache HBase (Cattell, 2010; Lóscio,2011).

A Google desenvolveu sua própria solução NoSQL, chamada BigTable, que é um sistema de armazenamento distribuído em larga escala (Sousa et al, 2011). A solução BigTable trabalha com outros pacotes da Google, tais como GFS (*Google File System*) para gerenciamento de informações e o *map/reduce* para distribuição dos dados (Sousa et al, 2011 e Lóscio, 2011).

O Cassandra é um banco de dados de armazenamento distribuído para o gerenciamento de grandes quantidades de dados espalhados por centenas de máquinas (Sousa et al, 2011). O Cassandra foi criado, inicialmente, para otimização de busca do *facebook.* Atualmente, o Cassandra é usado para dar suporte à replicação, detecção de falhas, armazenamento em *cache,* dentre outras funcionalidades do *facebook* (Lóscio et al, 2011). A vazão de operações de leitura e escrita pode crescer linearmente e novas máquinas são adicionadas sem nenhum custo ou interrupção para a aplicação (Sousa et al, 2011).

2.3.5. Modelo de Dados Orientado a Documentos

O modelo de dados Orientado a Documentos armazena uma coleção de documentos. Cada documento tem um conjunto de campos (chaves) e o valor do campo semelhante ao modelo de dados chave-valor (Cattell, 2010; Lóscio,2011).

Alguns bancos de dados que utilizam esse tipo de modelo de dados são: MongoDB, CouchDB (Cattell, 2010; Lóscio, 2011).

No CouchDB e MongoDB os documentos são armazenados e acessados como objetos *JavaScript Object Notation* (JSON). Operações de atualização são executadas sobre todo o documento e o CouchDB gerencia as alterações por meio de um identificador de revisão contido em cada documento. O CouchDB utiliza uma estrutura de arquivo baseada em árvores B+ para persistir os dados (Sousa et al, 2011 e MongoDB, 2013).

2.3.6. Modelo de Dados Orientado a Grafos

O modelo de dados orientado a grafos tem por finalidade instanciar dados que naturalmente podem ser estruturados na forma de um grafo (ANGLES, 2012). Uma rede social é um exemplo típico, onde temos usuários que se conectam entre si de varias maneiras (amigo, seguidor, etc.) e se conectam com outros objetos, tais como idéias, *posts*, produtos (curtir, seguir, etc.).

Os SGBDGs estão se tornando cada vez mais popular, por prover uma robusta solução para lidar com uma grande quantidade de dados onde as informações sobre a inter-conectividades ou a topologia dos dados são mais importantes, ou tão importantes quanto os dados propriamente dito (CASTELLTORT et al, 2013).

Ao invés de tabelas, colunas e relacionamentos um SGBDG é composto por nós, propriedades e relacionamentos. Para exemplificar podemos citar o e-commerce de livros, os nós podem ser livros, autores e editoras. Os livros podem ser descritos em propriedades como título, gênero, preço, número de paginas, etc. As propriedades dos autores podem ser nome, data de

nascimento, biografia, etc. Os relacionamentos estabelecem as relações entre os nós. Exemplo: Escrito_por, Editado_por ou É_autor_de.

Alguns exemplos de SGBDGs disponíveis no mercado que trabalham com essa tecnologia são:

- **DEX:** é um SGDBG adequado para grandes quantidades de dados e foi desenvolvido por pesquisadores da Universidade Politécnica da Catalunha. DEX é nativamente disponível para .Net, C ++, Python e Java, e para qualquer sistema operacional, mesmo Android e iOS. Ele também tem indexação nativa, que permite acesso rápido às estruturas de dados do gráfico. DEX oferece uma versão restrita para uso pessoal, mas não é open-source.
- **InfiniteGraph:** é um SGBDG distribuído orientado para apoiar os gráficos de grande escala, disponível em ambas as versões de licenças gratuitas e pagas. Destina-se a travessia eficiente das relações através de armazenamentos de dados em massa e distribuídos e sua linguagem é Java.
- **Neo4j** é um SGBDG de propriedade de código aberto totalmente transacional com mecanismo de persistência Java e fornece diferentes APIs para Ruby, Python e Java, com suporte para várias tecnologias de Web. Ele fornece suporte a transações ACID completos (Neo4j, 2014). Por estas características, Neo4j foi o SGBDG escolhido para o estudo de caso e para ele foi implementado a camada de segurança do controle de acesso.

Neste trabalho usaremos o banco Neo4j; a escolha do banco de dados se deu pelo fato de que os dados são persistidos na forma de grafo, o que torna a modelagem, de inúmeros sistemas de redes complexas, natural. Acreditamos que a habilidade do Neo4j em resolver problemas que demandam várias consultas, a implementação interna dos algoritmos de busca e o fato de ser *open source* podem contribuir com as pesquisas e estas auxiliarem o desenvolvimento do gerenciador de forma a torná-lo mais seguro.

A Figura 2.10, apresenta os componentes básicos de um SGDBG Neo4j: onde os nós são os vértices dos grafos, os relacionamentos são as arestas e as propriedades os atributos. Cada nó ou relacionamento da base de dados possui suas próprias propriedades que podem começar de

forma singular e crescer até acomodar todos os dados da base, podendo representar milhões de nós interconectados.

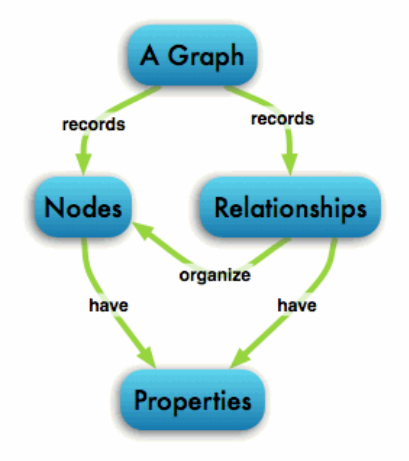

Figura 2.6: Modelo de dados do Neo4j (Neo4j , 2014)

Vickanair et al (2011), fez uma comparação entre o gerenciador de banco de dados NoSQL Neo4j e o relacional MySQL. Segundo o autor, uma vantagem em se usar o banco de dados Neo4j é o desempenho. Em pesquisas de textos, a base de dados Neo4j foi significativamente melhor que a base de dados relacional. Uma desvantagem bastante relevante é o aspecto de segurança, uma vez que o Neo4j não provê nenhum mecanismo de segurança (Vickanair et al, 2011),.

No manual do usuário do Neo4j capitulo Segurança no acesso ao Neo4j existe o aviso: *"O servidor Neo4j apresenta por padrão funcionalidades de script remoto que permite o acesso total ao sistema subjacente. Expor o seu servidor sem implementar uma camada de segurança representa uma vulnerabilidade de segurança substancial."* (Neo4j, 2014). Com essa nota, os autores do Neo4j alertam os usuários, mas delegam a eles a responsabilidade de resolver o problema.

Em aplicações em que se deseja uma maior segurança é aconselhável o uso de um Proxy como o Apache, por exemplo, podendo assim, controlar os IP ou faixa de IPs com acesso (Neo4j, 2013).

Outro mecanismo de segurança aconselhado no manual do Neo4j é a implementação de regras de segurança utilizando a linguagem Java. Após a implementação das regras deve ser feito o registro das classes desenvolvidas no arquivo *neo4j-server.properties* para que as políticas de segurança sejam aplicadas antes do acesso ao banco (Neo4j, 2014). Com o desenvolvimento dessas regras, os administradores podem exigir políticas de segurança mais refinadas, permitindo o controle de acesso controlado ao banco de dados. A dificuldade é que as regras ficam implementadas em códigos Java, a cargo do desenvolvedor, ficando dependentes da qualidade do desenvolvimento e manutenção dessas classes. O administrador da base de dados fica desprovido de ferramentas para gerenciar a base de dados adequadamente. Além disso, o arquivo é depositado no dispositivo de persistência (por exemplo, disco rígido) e pode ser facilmente manipulado sem que preocupações importantes, tais como, a privacidade das informações ou a própria conservação do arquivo de dados possam ser gerenciadas.

Outras estratégias disponíveis para garantir a disponibilidade e a confiabilidade no gerenciador de banco de dados Neo4j é o backup *online*, onde apenas uma instancia mestre é usada com o *backup* habilitado. Em caso de falhas, os arquivos de backups podem ser recuperados e utilizados na aplicação, ou no caso do *backup online* de alta disponibilidade, outra instância pode assumir automaticamente quando a instância mestre falhar (Neo4j, 2014).

Podemos utilizar *clusters* no Neo4j, onde temos um mestre (leitura/gravação) e um número de escravos (leitura). Em caso de falha nos escravos, estas instâncias são reiniciadas e o processamento não é interrompido. Por outro lado, se a falha for a instância mestre, um dos escravos assume o lugar da instância mestre (Neo4j, 2014).

Uma das principais características do Neo4j é manter um sistema transacional que respeita as tão conhecidas propriedades ACID, muito comuns em bancos de dados relacionais, na qual é dito que as transações devem ser atômicas, levar o banco de dados de um estado consistente a outro, manter o isolamento das transações e cuidar da durabilidade dos dados. Juntas, essas características formam um dos principais pilares que sustentam os sistemas de bancos de dados relacionais

2.3.7. Linguagem de consulta Cypher

O Neo4j suporta diversas linguagens de consultas para grafos na manipulação da sua base de dados, tais como Cypher, Gremlin, REST API, Lucene, entre outras (Neo4j, 2014). A linguagem de consulta que usaremos para manipular a base de dados Neo4j é o Cypher. A linguagem foi escolhida por ser uma linguagem relativamente simples, similar ao SPARQL ou SQL.

Com o Cypher podemos selecionar, inserir, atualizar ou excluir dados em uma base de dados gráfica. Mas não temos comandos para definição de esquema de grafos para a base de dados. Como o Neo4j não possui definição de esquema de dados, vamos propor no decorrer deste trabalho esta definição, e para isto usaremos os comandos do Cypher de instanciação dos nós na base de dados.

A Figura 2.7, apresenta uma comparação do grafo e a sintaxe da consulta Cypher. No grafo podemos ver o nó “a LIKES b”, ou seja o nó “a” está ligado ao nó “b” pelo relacionamento “LIKES”. No Cypher representamos os nós por parênteses “(a)”, o tipo do relacionamento por colchetes “[:LIKES]”.

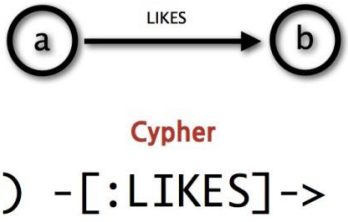

Figura 2.7: Cypher usando o relacionamento “LIKES” (Neo4j , 2014)

A estrutura da consulta da linguagem segue os comandos a seguir:

- MATCH: a forma mais comum de se obter os dados do grafo. Indica os nós, ou caminhos que devemos percorrer.
- RETURN: indica quais as propriedades a consulta deve retornar.

A Figura 2.8 apresenta alguns exemplos de consulta usando a linguagem Cypher. O primeiro exemplo mostra a sintaxe da linguagem para uma consulta simples que retorne todas as

propriedades de todos os nós da base de dados. O segundo exemplo é uma consulta simples que retorna todas as propriedades de todos os nós da base de dados e por fim a terceira consulta retorna todas as propriedades de todos os nós da base de dados que possui um relacionamento denominado “LIKES” com outro nó.

```
//sintaxe da consulta
//MATCH (node) RETURN node;

//consultar todas as propriedades no nó “a”
MATCH (a) RETURN a ;

//consultar todas as propriedades das pessoas que o nó “a” possui um relacionamento de “LIKES”
MATCH (a) –[:LIKES] -> (b)
RETURN a;
```

Figura 2.8: Exemplos de consulta utilizando Cypher

Para enquadrar os nós em grupos e conseguir fazer a distinção dos tipos de nós, por exemplo, Pessoas ou Empresas podemos nomeá-los por *labels*. A Figura 2.9, apresenta um exemplo de consulta utilizando os *labels*, no exemplo seria retornado todos os nós que se enquadram com o *label* “Pessoa”.

```
//consultar o nó com label “Pessoa”
//MATCH (node:label) RETURN node ou seja:
MATCH (a:Pessoa) RETURN a ;
```

Figura 2.9: Exemplos de consulta utilizando *label* (Cypher)

O comando CREATE é usado para instanciar novos nós. Os atributos do nó devem vir dentro de chaves “{ }”. O agrupamento do nó, em um *label,* é opcional. A Figura 2.10, apresenta um exemplo de criação do nó, agrupado ao *label* “PESSOA” com os atributos Nome e RG. O valor das propriedades do tipo string devem vir preenchidas entre aspas duplas (“ ”).

```
//sintaxe da criação do no
//create (node:label { Propriedade: “valor”, Propriedade: "valor"};

create (n:PESSOA {Nome: “Ana”, RG: “28.058.157-x”};
```

Figura 2.10: Exemplos de criação de nós (Cypher)

Semelhante ao SQL a restrição da consulta no Cypher pode ser feita com a cláusula WHERE, a diferença é que o Cypher é *case sensitive* tanto para o nome da propriedade quanto para o valor da mesma.

A Figura 2.11 apresenta um exemplo em que o resultado retorna os atributos Nome e RG dos nós cujo atributo Nome seja IGUAL a "Ana".

```
// Restringir a consulta para selecionar todos os nós com a propriedade "Nome" igual "Ana"
MATCH (a)
WHERE a.Nome = "Ana"
RETURN a.Nome, a.RG
```

Figura 2.11: Exemplos de consulta com restrições de retorno (clausula WHERE)

Para excluir ou alterar o valor da propriedade de um nó ou relacionamento, primeiramente temos que realizar a consulta que retorne o nó ou relacionamento para alteração ou exclusão e em seguida realizar a operação. O comando para realizar a alteração é o SET e para exclusão é o DELETE. A Figura 2.12 mostra um exemplo de alteração e um exemplo de exclusão. O primeiro exemplo é uma alteração (SET), neste exemplo esta sendo alterado a propriedade RG para "28.058.154-X" do nó com a propriedade Nome igual a "Ana". O segundo exemplo mostra a exclusão (DELETE) do nó com propriedade RG igual a "28.058.154-X" .

```
// alterar o RG do nós com a propriedade "Nome" igual "Ana"
MATCH (a)
WHERE a.Nome = "Ana"
SETa .RG = "28.058.154-X";

// excluir o nó com a Propriedade RG = "28.058.154-X"
MATCH (a)
WHERE a.RG = "28.058.154-X"
DELETE a;
```

Figura 2.12: Exemplos de alteração e exclusão utilizando Cypher

2.4. Considerações finais do capítulo

As limitações dos SGBDs relacionais para o gerenciamento de grande volume de dados semi-estruturados ou não estruturados, que necessitam de alta disponibilidade e escalabilidade, tem exigido uma atenção maior aos SGBDs NoSQL, considerado uma alternativa para o problema.

Este capítulo apresentou o referencial teórico deste trabalho, resumiu alguns conceitos básicos e as características dos SGBDs NoSQL. É grande a diversidade de aplicações que podem se beneficiar do modelo de dados orientado a gratos. Newman (2003) classificou esses modelos em quatro classes distintas: redes sociais; redes de informação; redes de tecnologia; e redes biológicas. Escolhemos o SGBDG, ou seja, a estrutura e o modelo orientado a grafo como objeto de estudo deste trabalho.

Depois de uma análise dos SGBDGs disponíveis foi utilizado o Neo4j para apoiar a implementação deste trabalho. A escolha recaiu sobre ele por ser um SGBDG *open source*, que respeita as tão conhecidas propriedades ACID e traz, nativo ao SGBDG, um diretório para configuração de *plugins*, permitindo que se implemente o modelo de segurança como um *plugin*. E por fim, foi apresentado a linguagem de consulta a grafos Cypher, linguagem utilizada pelo Neo4j similar ao SPARQL ou SQL.

3. Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados na área de segurança da informação de SGBDs relacionais e não somente relacionais. Os temas abordados tem inicio na seção 3.1, com o levantamento dos requisitos básicos para avaliar o controle de segurança de sistemas de computador. A seção 3.2, descreve um comparativo entre os SGBDs NoSQL e o motivo da escolha do SGBDG Neo4j para implementar o modelo de segurança. Em seguida a seção 3.3 apresenta um estudo dos metadados para serem aplicados no controle de acesso do SGBDG. Por fim, a seção 3.4 apresenta as considerações finais do capítulo.

3.1. Níveis de Segurança

Para esse trabalho a segurança em banco de dados é muito importante e por isso consultamos o *Trusted Computer System Evaluation Criteria* (TCSEC) que é um guia do Departamento de Defesa dos Estados Unidos (DoD, 1983), que estabelece requisitos básicos para avaliar o controle de segurança de sistemas de computador.

Dentre os requisitos mínimos considerados para a segurança de um sistema de computador está o controle de autorização de acesso a dados como *Discretionary Access Control* (DAC) que restringe o acesso a objetos baseado na identidade do usuário e/ou grupos aos quais pertencem (DoD, 1983). Segundo Yang (2009) o fato do mecanismo de controle de acesso DAC não impor qualquer restrição ao controle do fluxo de informação, deixa a informação vulnerável podendo disponibilizar informações a outros usuários, que não deveriam ter permissão de conhece-las.

Outro mecanismo de segurança citado no DoD (1983) é o *Mandatory Access Control* (MAC). Este tipo de mecanismo é baseado em regulamentações obrigatórias, ou seja, apenas os usuários administradores do sistema têm a autorização de implementar políticas de segurança para os demais usuários.

Uma alternativa para esses controles de acesso segundo Yang (2009) é o *Role-Based Access Control* (RBAC), onde os administradores concedem permissões de acordo com as funções de trabalho dentro uma empresa ou organização (Sandhu, 1998). O Privacy-Aware baseada em

função Access Control (P-RBAC) (Martino et al, 2009) é uma extensão do RBAC para incorporar noções de privacidade. Essas políticas são baseadas nas regras de privacidade como: tipo de dados, operações com dos dados, condições e restrições de acesso (Martino et al, 2009). No modelo básico P-RBAC a condição é escrever usando operadores relacionais e variáveis de contexto, complexas e difíceis de gerenciar.

Os bancos de dados relacionais possuem esses mecanismos de segurança nativos em seus gerenciadores de Banco de Dados. Portanto, o acesso dos usuários que estejam utilizando as aplicações é controlado automaticamente pelo mecanismo de controle de acesso existentes nos gerenciadores relacionais, enquanto a maioria dos SGBDs NoSQL deixa este controle para aplicação

O trabalho de Colombo e Ferrari (Colombo e Ferrari, 2015) propõe uma abordagem para o reforço do modelo de controle de acesso baseado em papéis MongoDB, incluindo conceitos de privacidade. O modelo RBAC (Role-Based Access Control) do MongoDB foi estendido com os conceitos de uso e mecanismos de execução conexas para regulamentar o acesso no nível do documento com base em políticas de propósito e papel à base. Um monitor de aplicação, chamada Mem (MongoDB Monitor de Execução), foi concebido para implementar o modelo reforçada prevista. Embora os autores a melhorar o controle de acesso, o foco deste trabalho é especificamente o MongoDB, que é um banco de dados NoSQL orientado a documentos. Nosso trabalho propõe um modelo para bancos de dados orientados para o gráfico e o objetivo do modelo é orientar a implementação de mecanismos de controle de acesso para bancos de dados diferentes, nesta categoria, não uma em particular.

Accumulo (Accumulo, 2016) é um armazenamento de chave-valor que apoia as políticas de controle de acesso à base de células, onde uma célula representa uma linha e uma combinação coluna. Neste mesmo sentido, Kul-karni (Kulkarni, 2013) propõe um modelo de controle de acesso para sistemas de valores-chave, que permite a especificação de políticas ao nível da keyspace, família de colunas, linhas e colunas.

Em (Kulkarni, 2013) implementações TTWO foram definidos. No primeiro caso, o controle de acesso é inte-ralado modificar o código fonte de Cassandra, enquanto no controle último, o acesso é executada por um módulo de software externo que interage com os dados

armazenados. Além de ser específico para banco de dados Cassandra, este modelo ainda é limitada porque as permissões são apenas para leitura e escrita. O nosso modelo é mais completa, proporcionando permissões como alterar e apagar, bem como a criação da estrutura de nodos.

Zhu et al (Zhu et al, 2009) desenvolveu um modelo de controle de acesso obrigatório prático para bancos de dados XML, que são um tipo anterior de bancos de dados orientados a documentos. O modelo é baseado em regras, como regra de acesso de leitura, a regra de acesso escrever e regras atribuições do rótulo. No trabalho de André Calil e Ronaldo dos Santos Mello (Calil e Mello, 2012) foi criada uma camada relacional sobre o banco de dados não relacional SimpleDB, com o objetivo de se criar uma interface SQL que transforma as requisições feitas em SQL para requisições não relacional do SimpleDB. Esta camada foi denominada SimpleSQL e faz o mapeamento de quatro operações básicas do banco de dados relacional INSERT, UPDATE, DELETE e SELECT para o SimpleDB.

A camada de segurança é independente utilizando o próprio gerenciador de banco de dados NoSQL (nesse caso, o Neo4j) controlando o acesso aos dados persistidos no banco de dados (nesse caso, no Neo4j).O fato do de ser baseado em algumas regras e ter apenas operações de Data Manipulation Language (DML), constitui uma limitação para estes ambas as obras (Zhu et al, 2009) (Calil e Mello, 2012). Além disso, foram definidos para bancos de dados orientados a documentos. Nosso modelo fornece também as operações de Data Definition Language (DDL) e a definição de um esquema para orientada para o gráfico de banco de dados, com base na estrutura de nós.

O gerenciamento de usuários e seus privilégios podem tornar-se uma tarefa árdua para o DBA, dependendo do número de nós e usuários de SGBDG. Assim, complementarmente às obras anteriores descritos (Colombo e Ferrari, 2015)(Accumulo, 2016)(Kulkarni, 2013)(Zhu et al, 2009) (Calil e Mello, 2012), o modelo de controle de acesso apresentado neste artigo fornece permissões baseadas em grupos. Os usuários podem ser associados a grupos de usuários e as permissões podem ser concedidas apenas uma vez, ao grupo de usuários.

Um trabalho recente incide sobre fraquezas adicionais de armazenamentos de dados NoSQL, tais como a confidencialidade dos dados que armazenam (por exemplo, o trabalho de Yubin et al, 2013), que propõe uma solução de armazenamento de dados e protocolo de consulta

com base na criptografia para preservar a privacidade dos proprietários e consulta de dados usuários), mas não temos conhecimento de outras propostas para a integração eficaz de aplicação de controle de acesso consciente de privacidade em sistemas NoSQL, especialmente para bancos de dados orientados para o gráfico.

3.2. Comparativo entre SGBDs NoSQL

A segurança dos dados é um dos maiores desafios enfrentados pelos SGDBs NoSQL nos dias de hoje. O trabalho de Okman et al (2011) aborda a segurança em dois populares SGBDs NoSQL, Cassandra e MongoDB. Os principais problemas apontados e comuns a ambos os sistemas, incluem a falta de suporte a criptografia dos arquivos de dados, fraca autenticação tanto entre o cliente e os servidores quanto entre membros do servidor. Okman et al (2011), deixa claro que gerações futuras de tais SGBDGs precisam de um considerável desenvolvimento, a fim de fornecer ambiente seguro.

Alguns anos mais tarde Zahid et al (2014), volta a fazer uma análise comparativa na segurança dos SGBDs NoSQL MongoDB, Redis, CouchDB, Cassandra, HBase e CouchbaseServer. Foram analisados os aspectos de autenticação, controle de acesso, configurações seguras, criptografia dos dados e auditoria, conforme podemos ver na Tabela 3.1, considerado como controle Baixo os SGBDs que não apresentam a características, Médio ao que estão introduzindo ao SGBDG algum controle de segurança e Alto quando possuem controle de segurança.

Na análise da Tabela 3.1, podemos verificar que a maioria dos SGBDs NoSQL comparados, tem implementado a parte do controle de acesso em um nível básico, onde todos os direitos de leitura, escrita e alteração são concedidas a um usuário default. Estas bases de dados não possuem suporte para os usuários concederem permissões distintas a diferentes usuários, com exceção ao MongoDB e HBase que possuem uma granularidade de permissão mais baixa. O MongoDB foi implementado com suporte ao RBAC (*Role-Based Access Control*), onde os administradores concedem permissões de acordo com as funções de trabalho dentro de uma empresa ou organização. E o HBase foi implementado com suporte ao ACL (Access Control List) (HBase,

2015). O Neo4j possui *plugins* que podem ser configurados ao Neo4j e habilitado para autenticação do usuário, mas não possui restrições de acesso destes usuários, uma vez conectado a base de dados o usuário pode incluir, alterar, excluir ou selecionar qualquer nó dentro da base de dados.

Tabela 3.1: Análise comparativa entre vários SGBDs NoSQL (adaptada de Zahid et al. 2014).

Itens avaliados	MongoDB	Redis	CouchDB	Cassandra	HBase	CouchbaseServer	Neo4j
Autenticação	Média	Baixa	Média	Média	Média	Média	Média
Controle de Acesso	Alta	Baixa	Baixa	Baixa	Média	Baixa	Baixa
Configurações Seguras	Médio	Baixa	Baixa	Baixa	Baixa	Baixa	Baixa
Criptografia	Médio	Baixa	Média	Média	Baixa	Baixa	Baixa
Auditoria	Baixa	Baixa	Média	Média	Média	Média	Baixa

Com base na comparação entre o SGBD relacional MySQL e o SGBD NoSQL Neo4j Vicknair et al (2010), considera prematuro usar o SGBDG Neo4j em um ambiente de produção, e a falta de segurança é um dos requisitos fundamentais para esta decisão.

O trabalho de Angles R. (2012), faz uma comparação entre SGBDGs. Analisando a modelagem dos dados, comparando a representação de nós, atributos e relacionamentos em nível de esquema e instância, conforme mostrado na Tabela 3.2.

Tabela 3.2: Comparação da modelagem dos dados (Angles, R., 2012).

SGBDG	Esquema			Instância					
				Nós			Relações		
	Tipos de Nós	Tipos de Atributos	Tipos de Relações	ID	Atributos Simples	Atributos Complexos	ID	Relações Simples	Relações Complexas
AllegroGraph					x			x	
DEX	x		x	x	x		x	x	
Filament					x			x	
G-Store					x			x	
HyperGraphDB	x		x		x			x	x
InfiniteGraph	x		x	x	x		x	x	
Neo4j				x	x		x	x	
Sones					x			x	x
vertexDB					x			x	

Na Tabela 3.2, o nível de esquema encontrado nestes modelos suporta a definição de tipos de nós, propriedades e relacionamentos. O nível de instância reconhece uma entidade como um nó que contém propriedades (atributos). Na análise Tabela 3.2, o suporte do modelo em nível de instância foi baseado em três características distintas:

- ID: cada nó tem seu identificador único na instância (Object-ID);
- Atributos simples: valores dos atributos primitivos do nó, exemplo nome.
- Atributos complexos: – valores complexos formado por um conjunto de informações.

O mesmo tipo de análise acontece com os relacionamentos:

- ID: cada relação tem seu identificador único na instância (relation-ID).
- Relações simples: um simples relacionamento na instância, representado por nó-relação-nó.
- Relações complexas: é um relacionamento com semânticas especiais, por exemplo, derivação, herança.

Os atributos simples nos nós e relações são aceitos por todos os modelos analisados, o motivo é que isso é o básico que se espera para representar grafos. A introdução de conceitos de orientação a objetos (por exemplo IDs de objetos) reflete o uso de APIs.

O fato da análise de segurança do Neo4j, ter o aspecto de controle de acesso baixo, sem restrições de acesso para diferentes perfis, foi um fator de motivação na escolha desse SGBDG para o estudo de caso. O modelo de segurança, traz ao SGBDG Neo4j a definição de esquema, deixando a base mais consistente e homogenia e faz restrições de acesso a diferentes tipos de perfis de usuários.

3.3. Metadados

Da literatura, podemos definir que metadados são dados sobre os dados, podendo ser divididos em duas categorias básicas: Metadados Técnicos e Metadados de Negócios (Cerqueira, 2012). Segundo Cerqueira (2012), a utilização de um repositório de metadados possibilita uma visualização integrada de todo o ambiente de dados da empresa.

A ferramenta Yellowfin BI (solução de *Business Intelligence Real-Time)*, possui uma camada de Metadados que permite aos desenvolvedores entenderem o significado dos dados e o contexto do negócio (Yellowfin, 2012). Nela, os administradores podem mapear os dados de acordo com um esquema de classificação, objetivando a criação de uma visão das informações de negócio. Uma das funcionalidades do Yellowfin é o controle de acesso aos usuários e o controle de permissão de acesso aos relatórios e dados gerados (Yellowfin, 2012). Nesse caso, não há conexão com banco de dados não relacionais (NoSQL).

A camada de dados para o controle de acesso à base de dados do gerenciador Neo4j é baseada em metadados para sua construção.

3.4. Considerações finais do capítulo

Este capítulo apresentou os trabalhos correlatos na área de segurança da informação utilizadas no SGBDs relacionais e não somente relacionais.

Primeiramente apresentados os tipos de regras de autorização e a regra RBAC (Role-Based Access Control), onde os administradores concedem permissões de acordo com as funções de trabalho dentro uma empresa ou organização.

Analisando a segurança dos SGBDs NoSQL com Okman et al (2011), Zahid et al (2014) e Vicknair et al (2010) , reafirmamos a necessidade do amadurecimento da segurança nos SGBDs NoSQL.

Com o trabalho de Angles, R. (2012) pudemos comparar diversos SGBDGs e escolher o Neo4j como o SGBDG usado para implementar o modelo de segurança. E por fim, analisamos o controle de acesso, mapeados por metadados, estudando uma solução BI que utiliza um metadados para determinar o controle de acesso.

4. Modelo de Segurança para Controle de Acesso

Como parte de um modelo de segurança, propõe-se a representação de um controle de acesso com o apoio de metadados e, assim, permitir que se garanta que usuários não autorizados não possam ter acesso e modificar os dados do SGBDG.

Os temas abordados têm início na seção 4.1 com a descrição dos metadados utilizados para a autenticação do usuário no controle de acesso ao SGBDG. A seção 4.2 descreve as atribuições de privilégios concedidas individualmente ao usuário ou herdadas de um grupo de usuário. A seção 4.3 descreve os privilégios para criação, alteração e exclusão do esquema do BDG. A seção 4.4 descreve os privilégios para manipulação dos nós como inclusão, alteração, exclusão de seleção dos nós. E por fim, a seção 4.3 apresenta as considerações finais deste capítulo.

4.1. Contas de Acesso

Quando vários usuários utilizam um banco de dados, é provável que a maioria desses usuários não seja autorizada a acessar todas as informações disponíveis no banco de dados, por isso a necessidade de autenticação de usuário e controle de acesso à base de dados.

É comum os SGBDs relacionais fornecerem aos usuários, ou grupo de usuários, consultas/alterações protegidas por senhas, utilizadas para acessar o banco de dados. (ELMASRI e NAVATHE, 2005).

Nos bancos de dados relacionais o administrador do banco de dados (do inglês, *DataBase Administrator* - DBA) é a autoridade principal no gerenciamento de um SGBD. As responsabilidades do DBA incluem a concessão de privilégios a usuários que precisam utilizar o sistema e a classificação de usuários de acordo com a política da organização (ELMASRI e NAVATHE, 2005).

Para conceder ou retirar permissões de um usuário ou grupo de usuários os SGBDs relacionais possuem a linguagem de controle dos dados,do inglês *Data Control Language* - DCL, com comandos do tipo:

- GRANT: concede privilégios a um usuário ou grupo de usuários;
- REVOKE: remove os privilégios do usuário ou grupo de usuários;
- DENY: nega um privilégio a um usuário ou grupo de usuários;

Similar ao modelo utilizado pelos SGBDs relacionais, o modelo de controle de acesso sugere a criação de um usuário para administrar o SGBDG (DBA), responsável por criar novas contas de usuários, conceder ou revogar privilégios para contas individuais ou grupo de usuários.

A criação da conta de usuário deve seguir a estrutura definida pelo metadados de segurança de usuário denominado Meta-Usuário.

A definição da estrutura de um nó de usuário é feita pela tripla N, L, A, como segue: G = {N,L,A}, onde N é o nó, L é a identificação do *label*, definido em maiúsculo por USUARIO e A é a coleção de atributos, tais como identificador, nome, senha, data da inclusão e o usuário que criou o nó na base de dados, ou seja, o usuário proprietário do nó. No processo de autenticação do usuário os mesmos serão identificados no BDG pelo *label* USUARIO.

A Figura 4.1 mostra o esquema do metadados Meta-Usuário e também um exemplo de instância. O usuário com nome “Usuario1” é cadastrado no BDG com os atributos identificador, nome, senha, data de cadastro e proprietário.

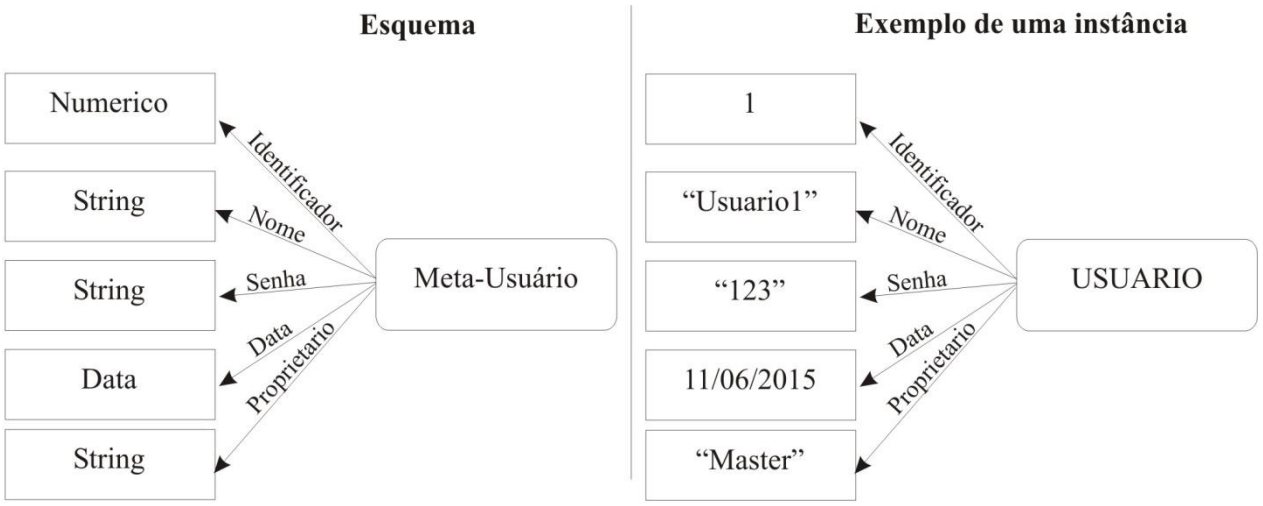

Figura 4.1: Metadados de autenticação de usuários.

O gerenciamento dos usuários e seus respectivos privilégios, pode se tornar uma tarefa árdua para o DBA, dependendo do número de esquema de nós e usuários do SGBDG. Para resolver este tipo de problema podemos associar os usuários a um grupo de usuários e conceder os privilégios uma única vez ao grupo de usuários.

A criação do grupo de usuário deve seguir a estrutura definida pelo metadados de segurança de grupo, denominado Meta-Grupo.

A definição da estrutura de um nó de grupo é feita pela tripla N, L, A, como segue: G = {N,L,A}, onde N é o nó, L é a identificação do *label*, definido em maiúsculo por GRUPO e A é a coleção de atributos, tais como identificador e nome.

Para que um usuário possa herdar os privilégios de um grupo de usuários, relacionamos o usuário (Meta-Usuario) ao grupo (Meta-Grupo) como podemos ver na Figura 4.2.

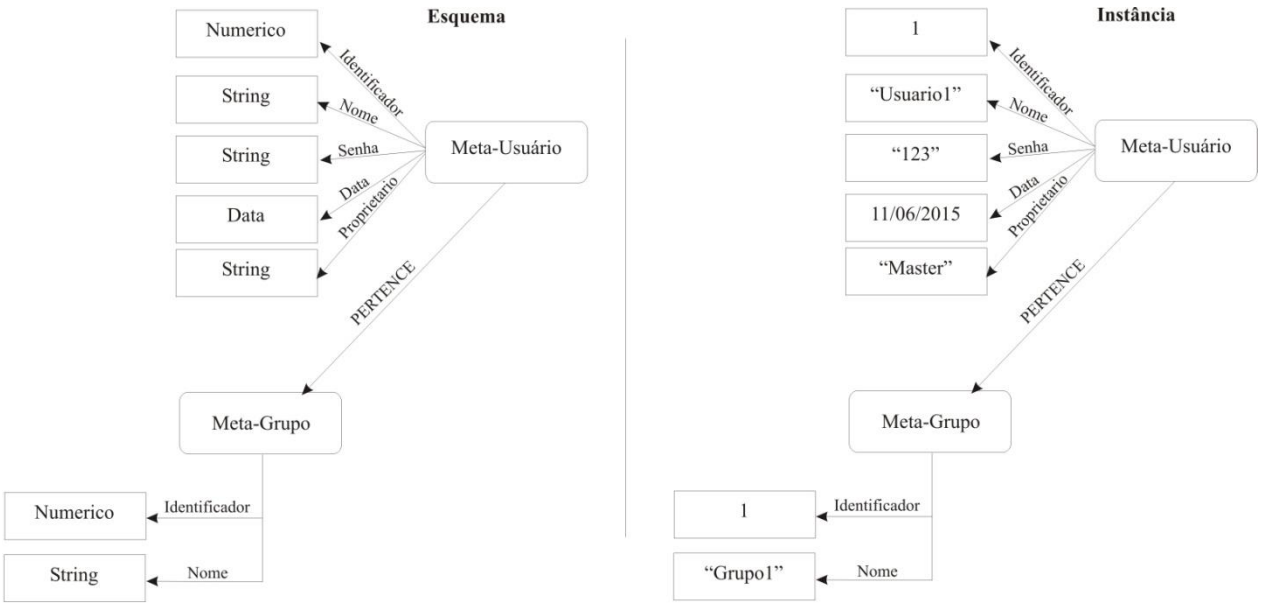

Figura 4.2: Metadados de grupo de usuários.

4.2. Atribuição de Privilégios

No modelo de controle de acesso os privilégios são cadastrados no BDG conforme a estrutura definida pelo metadados de segurança de operação, denominado Meta-Operação.

A definição de um nó de operação é feita pela tripla N, L, A, como segue: G = {N,L,A}, onde N é o nó, L o *label*, definido em maiúsculo por OPERACAO e A é a coleção de atributos, tais como, identificador e o tipo da operação permitida, que pode ser CREATE, DROP, ALTER, INSERT, DELETE, UPDATE ou SELECT.

O usuário pode receber cada um dos privilégios individualmente ou pode herdar esses privilégios de um grupo de usuários, ao qual ele pertence e que possuem essas permissões de acesso. Para isso, relacionamos o usuário (Meta-Usuario) ou grupo (Meta-Grupo) à operação (Meta-Operação) desejada conforme mostra a Figura 4.3 Para revogar o privilégio do usuário removemos este relacionamento.

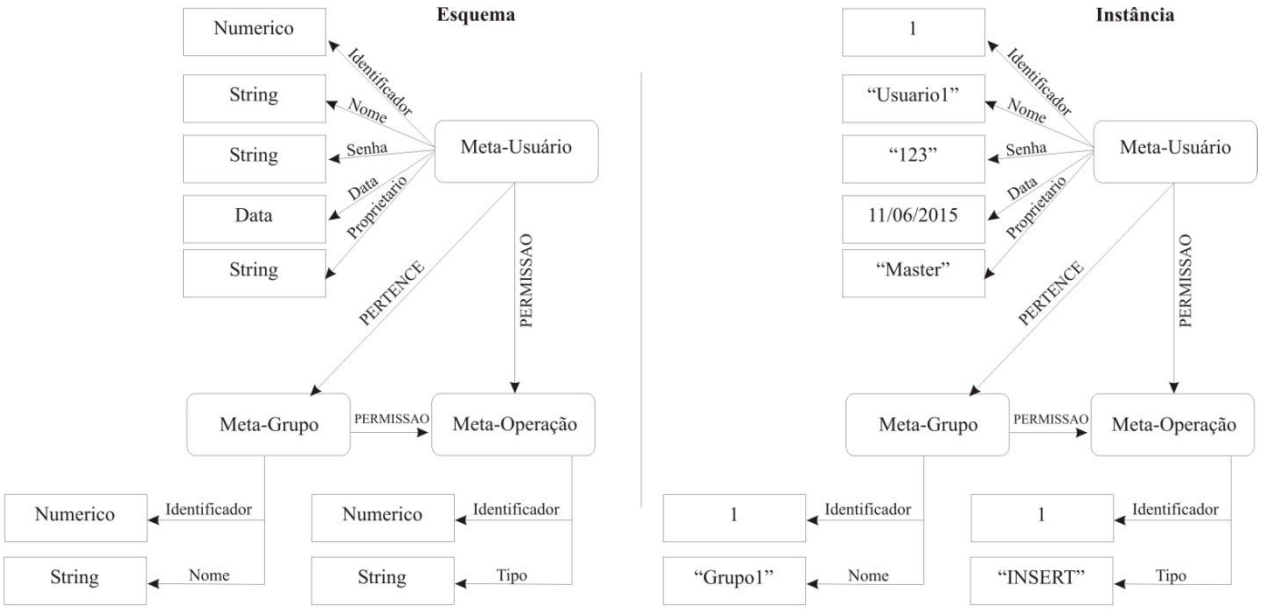

Figura 4.3: Metadados para concessão de privilégios a usuários ou grupo de usuários.

O modelo de segurança propõe a definição das estruturas dos grafos gerenciados pelos usuários. Para tanto, devem ser definidos as estruturas dos nós, que consistem nos atributos possíveis de cada nó, cujo metadados é donominado Meta-Nó.

A definição da estrutura de nó é feita pela tripla N, L, A, como segue: G = {N,L,A}, onde N é o nó, L o *label*, definido em maiúsculo por METANO e A é a coleção de atributos como o *label* e o proprietário do nó. O *label* que deve ser único no BDG.

Uma estrutura de nó pode ter um ou mais atributos definidos no metadados de segurança, denominado Meta-Atributo. A representação do grafo do atributo é feita pela tripla N, L e A, como segue: G = {N,L,A}, onde N é o nó, L o *label*, definido em maiúsculo por ATRIBUTO e A, a coleção de atributos, tais como, o identificador do atributo, nome do atributo, tipo de dados e indicação de obrigatoriedade.

A Figura 4.4 mostra o modelo de controle de acesso completo, onde o usuário pode estar associado diretamente a uma ou mais permissões ou a um grupo de usuário que possui a permissão. A permissão está associado à estrutura de Meta-Nó. Cada estrutura de Meta-Nó pode ter um ou mais atributos associados.

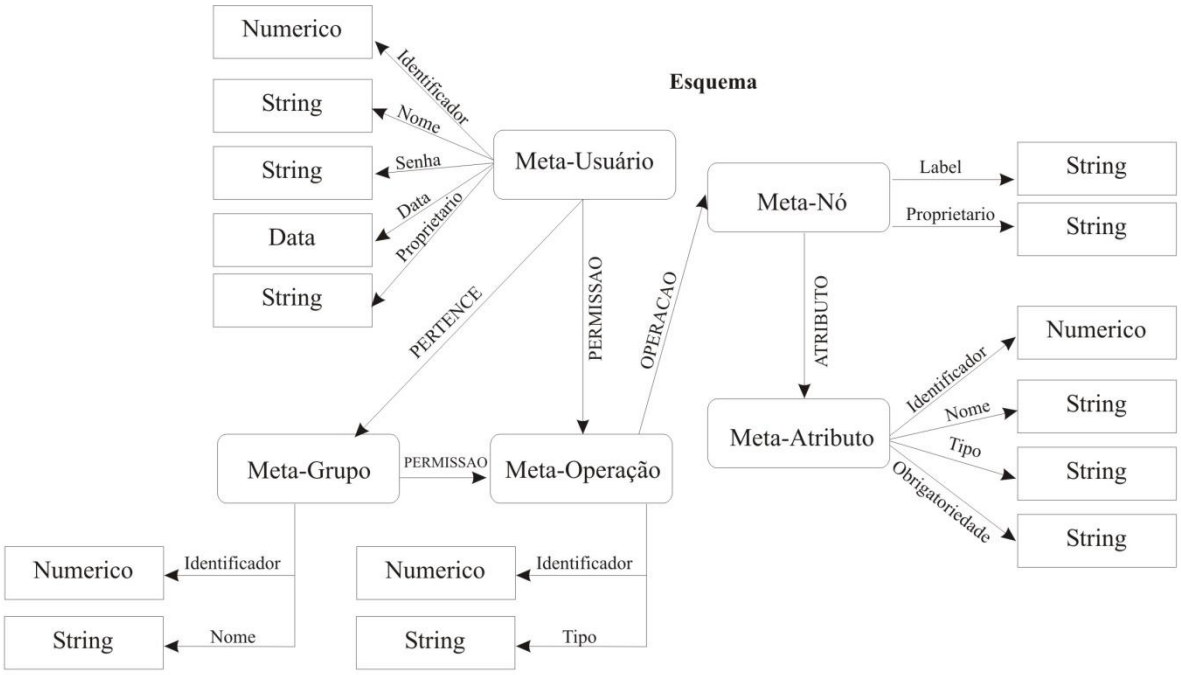

Figura 4.4: Modelo de controle de acesso para SGBDG.

O controle de acesso do usuário criado é baseado na concessão e na revogação de privilégios. Vamos considerar os privilégios no contexto do SGBDG em dois níveis: a definição do esquema do BDG e manipulação dos dados em si.

A definição do esquema do BDG está associada a privilégios de acesso de operações da linguagem de definição de dados,do ingles *Data Definition Language* - DDL, a serem abordados na seção 4.3. Privilégios relacionados a manipulação de dados, do inglês *Data Manipulation Language* - DML, são apresentados na seção 4.4.

4.2.1. Privilégios relacionados a operações de DDL

Os elementos de um esquema de um BDG são os tipos das entidades representadas pelos nós e os tipos de relacionamentos (ANGLES e GUTIERREZ, 2008).

Os privilégios de DDL deverão ser definidos sobre estes elementos de um esquema BDG de acordo com as seguintes operações:

- CREATE: permite que o usuário ou grupo de usuários receba a permissão de criar esquemas de nós. Com esta permissão o usuário poderá definir um *label*, único no BDG, os atributos que este nó poderá ter e a indicação do atributo ser obrigatório na inclusão.

- DROP: permite que o usuário ou grupo de usuários receba a permissão de excluir um esquema de nó. O usuário só poderá excluir o esquema de um nó se esse esquema não tiver instâncias criadas no momento da exclusão.

- ALTER: permite que o usuário ou grupo de usuários receba a permissão de alterar os atributos de um esquema de nó. O usuário não poderá alterar obrigatoriedade de um atributo se já existirem instâncias criadas que não definiram o atributo no momento da criação dessa instância. Dessa forma, a inclusão de novas instâncias terão esse atributo definido como obrigatoriedade opcional.

A Figura 4.5 mostra um exemplo de instância do modelo de controle de acesso representado na Figura 4.4 onde o "Usuario1" possui a permissão de "CREATE" e criou um esquema de nó "PESSOA" que possui dois atributos "Nome" e "Endereço". Durante a instaciação dos nós com o label "PESSOA", o atributo "Nome" é do tipo de dados *string* e obrigatório, enquanto o atributo "Endereço" é do tipo de dados *string* e opcional.

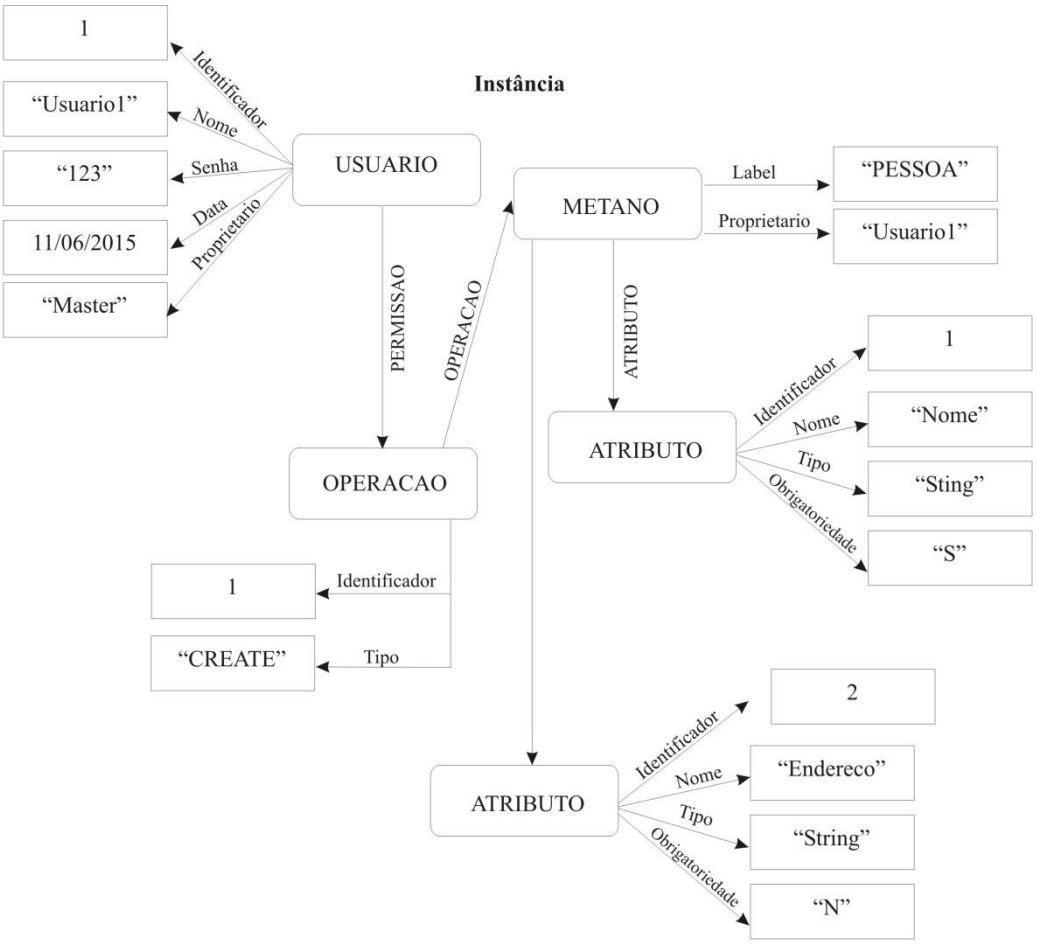

Figura 4.5: Instância do modelo de controle de acesso (Figura 4.4) para uma operação de DDL - CREATE.

4.2.2. Privilégios relacionados a operações de DML

Uma instância de um BDG contém as entidades concretas representadas pelos nós rotulados por um dos nomes de tipo de entidades definido na criação do banco de dados de acordo com um esquema de grafo, bem como, as relações representadas pelos relacionamentos rotulados por nomes de acordo com esse mesmo esquema (ANGLES e GUTIERREZ, 2008).

As manipulações típicas de um BDG são as transformações no grafo como caminhos, vizinhança, sub-grafos, nós pais, conectividades e estatísticas do grafo (diâmetro, centralidade,

etc.) (ANGLES e GUTIERREZ, 2008). Isso se dá através das operações de inclusão, alteração ou exclusão de nós e relacionamentos.

A concessão de uma permissão de DML ao usuário deve seguir a estrutura definida pelo metadados de segurança, de acordo com as seguintes operações:

- INSERT: permite que o usuário ou grupo de usuários receba a permissão de inserir nós no BDG. Os usuários só poderão inserir nós com os *labels* e atributos já definidos no esquema do BDG, que ele (ou grupo a que pertence) tenha privilégio de INSERT. Por exemplo: se um usuário possui permissão de INSERT (Meta-Operação) em uma esquema de nós com o *label* "PRODUTO" (Meta-Nó) e o produto possui os atributos Nome – Obrigatório e Marca – Opcional (Meta-Atributos), este usuário possui apenas o privilégios de inserir novos nós com o *Label* PRODUTO e respeitar os atributos definidos no esquema, sendo o Nome um atributo obrigatório.

- UPDATE: permite que o usuário ou grupo de usuários receba a permissão para alterar os dados dos atributos dos nós de acordo com os atributos definidas para o *label* desse nó específico na estrutura do BDG, sendo que o privilégio de UPDATE deve ter sido dado ao usuário ou ao grupo de usuários ao qual ele pertença. O usuário não poderá alterar o atributo para um valor nulo caso o esquema o defina como obrigatório.

- DELETE: permite que o usuário ou grupo de usuários receba a permissão de excluir nós que detenham o mesmo *label* daqueles para os quais o usuário tenha recebido privilégio de DELETE.

- SELECT: permite que o usuário ou grupo de usuários receba a permissão para ler todos os nós que detenham o mesmo *label* daqueles para os quais o usuário tenha recebido privilégio de SELECT.

A Figura 4.6 mostra um exemplo de instância do modelo de segurança representado na Figura 4.4 onde o "Usuario1" possui a permissão de "CREATE" e "INSERT" para o esquema de nó "PESSOA".

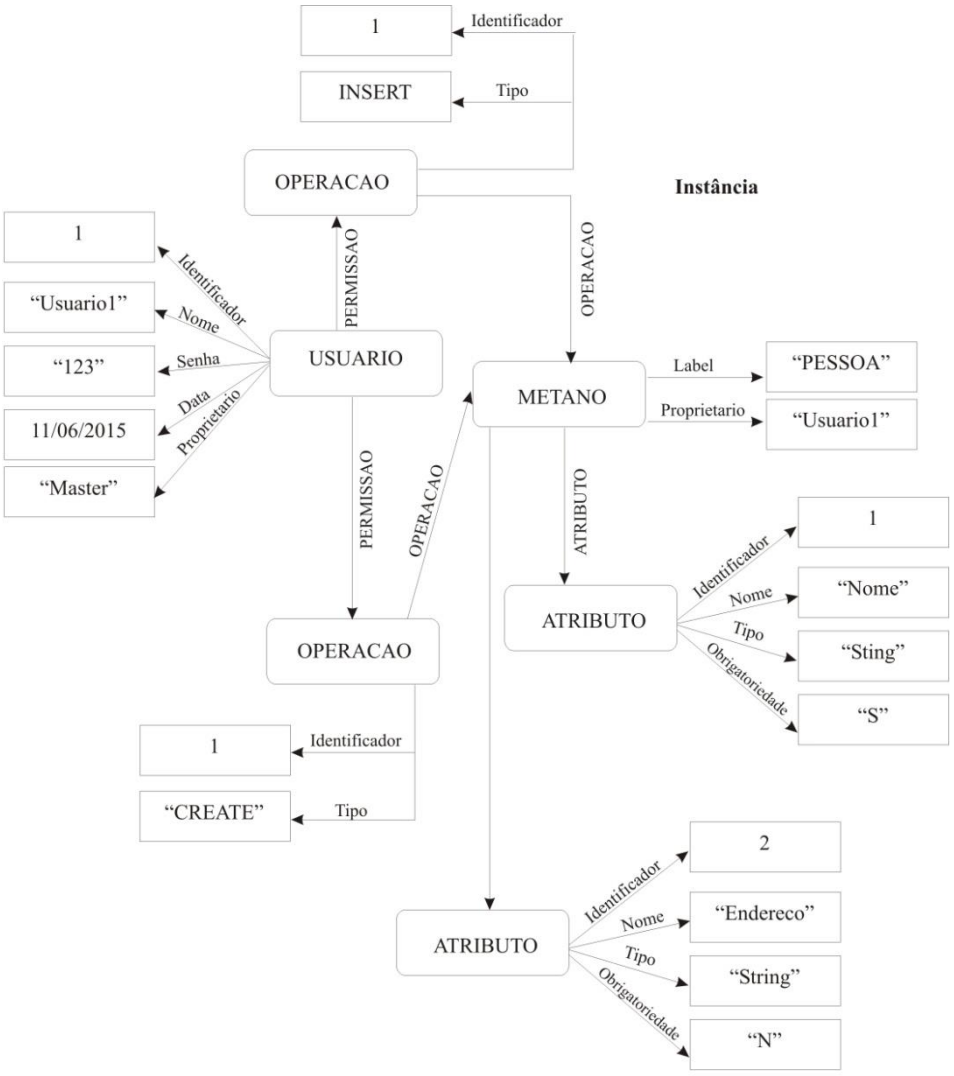

Figura 4.6: Instância do modelo de controle de acesso (Figura 4.4) adicionando a operação de INSERT.

4.3. Considerações finais do capítulo

Este capítulo apresentou o modelo de segurança, com foco no controle de acesso, para SGBDGs. O modelo presume a necessidade da representação de metadados para garantir que usuários não autorizados não possam acessar ou modificar dados.

Foram criados cinco estruturas de metadados para o controle de acesso ao SGBDG:

- Meta-Usuário: usuários que possui permissão de acesso ao BDG.
- Meta-Grupo: agrupa os usuários em grupos com as mesmas necessidades e mesmos tipos de permissões.
- Meta-Operação: todos os privilégios do BDG, create, drop, alter, insert, delete, update ou select.
- Meta-Nó: nome e proprietário da estrutura de nó.
- Meta-Atributo: atributos que pertencem à estrutura de nó.

Os metadados são relacionados de acordo com as permissões concedidas para os usuários. Um usuário (Meta-Usuário) pode receber privilégios individualmente ou herdar esses privilégios de um grupo de usuários (Meta-Grupo). Os privilégios do BDG estão relacionados as operações (Meta-Operação) que os usuários podem realizar no BDG.

Ao conceder o privilégio de CREATE para um usuário, ele passa a ter permissão de criar no BDG esquemas de nós (Meta-Nó). A cada esquema de nó criado é definido quais os atributos (Meta-Atributo) este esquema de nó deve ter, ou seja, quais as propriedades que cada instância do esquema de nó deve ter.

Quando um usuário cria um esquema de nó ele passa a ser o proprietário do nó, herdando automaticamente todas as permissões possíveis para este nó. O DBA poderá conceder permissões de DML para outros usuários criar, alterar ou excluir instancia de nós com a estrutura definida.

Para instanciar um nó, o usuário deve ter além da permissão de INSERT ao esquema de nó, respeitar os tipos de dados e obrigatoriedade do atributo (Meta-Atributo) pertencente a este esquema.

Um usuário apenas conseguirá excluir esquemas de nós se não existirem instancia do esquema de nó no BDG. O mesmo acontece para a alteração de um atributo que era opcional para obrigatório.

5. Estudo de Caso

O estudo de caso que será tratado implementa o modelo de segurança para controle de acesso ao SGBDG Neo4j, aplicado a um *website* de notícias. O motivo da escolha do SGBDG Neo4j foi por ser um SGBDG *open source,* escrito em Java, contemplando as propriedades ACID, mas com controle de acesso baixo, sem restrições de acesso a diferentes perfis.

O fato do Neo4j possuir nativo ao SGBDG um diretório para configuração de *plugins*, facilitou a implementação do modelo de controle de acesso como um *plugin*.

Este capítulo tem início na seção 5.1 que descreve o sistema utilizado para implementar o estudo de caso. A seção 5.2 descreve as ferramentas utilizadas e algumas considerações gerais da implementação para o desenvolvimento do modelo de controle de acesso. Esta seção está divida em três outras subseções: a subseção 5.2.1 contém a parte do modelo de segurança relacionada à autenticação do usuário no controle de acesso ao SGBDG, a subseção 5.2.2 descreve a criação, alteração e exclusão do esquema do BDG (DDL) e por fim a subseção 5.2.3 descreve o controle da manipulação dos nós como inclusão, alteração, exclusão de seleção dos nós (DML). A seção 5.3 descreve a validação do modelo de segurança com foco no controle de acesso com as evidencias de testes. Por fim, a seção 5.4 traz as considerações finais, discutindo os resultados da aplicação do modelo de segurança em um SGBDG.

5.1. Descrição do Estudo de Caso

O estudo de caso foi aplicado em um *website* de noticias, onde alguns usuários autorizados podem descrever em poucas linhas uma reportagem e dar um título a ela, e outros usuários comentar esta publicação.

As permissões neste *website* são bem diversificadas alguns usuários podem ter a permissão de publicar no site novas reportagens e realizar comentários sobre as reportagens publicadas, ou então ter permissão apenas de comentar as reportagens, ou apenas consultar as mesmas. A exclusão dos comentários também é controlada e apenas usuários com permissão podem excluir um comentário ou uma reportagem publicada.

Para um melhor entendimento do *website*, a Figura 5.1 apresenta a esquema relacional do estudo de caso. O diagrama é composto pelas entidades: usuário (USUARIO), reportagens (POST) e comentários (COMMENTS).

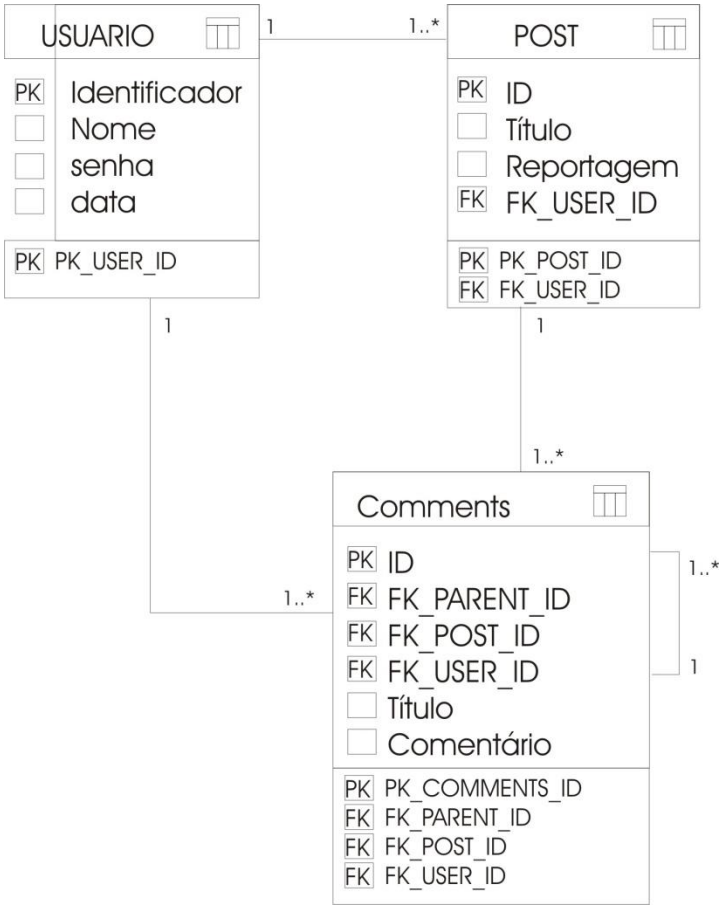

Figura 5.1: Esquema Relacional

5.2. Implementação do Estudo de Caso

Para estudo de caso o modelo de controle de acesso foi implementado no SGBDG Neo4j versão 2.1.3. O Neo4j suporta várias linguagens de consulta para manipular dados na base de dados (por exemplo, Cypher, Gremlin, REST API, Lucene, etc.) (SRINIVAS e NAIR, 2015).

Como abordado na seção 4.2 (Figura 4.2.2), o modelo de controle de acesso consiste em um metadados que é um grafo. Decidimos na instanciação dos nós desse grafo utilizar a

linguagem Cypher, apresentada na seção 2.3.8, devido à sua simplicidade e similaridade com SQL (Structured Query Language) e SPARQL (Protocolo Simples e RDF Query Language). Dentre os comandos do Cypher, os mais usados na instanciação do grafo de metadados foram os comandos básicos create, delete e set.

A funcionalidade do servidor Neo4j pode ser estendido pela adição de plugins. Plugins são códigos especificado pelo usuário que amplia os recursos do SGBDG, nós, ou de relacionamentos. Nossa implementação foi feita como uma extensão Neo4j, configurada no diretório plugin. A Figura 5.2 mostra o fluxo de dados através da implementação do modelo de controlo de acesso.

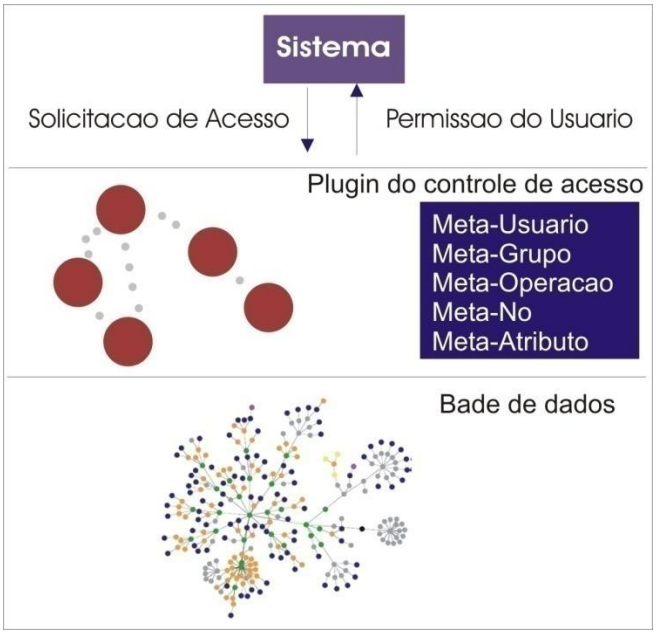

Figura 5.2: Fluxo de dados do controle de acesso

Na Figura 5.2, representa a interação de acesso do usuário com a base de dados do SGBDG. O solicitar acesso ao SGBDG o plugin de controle de acesso, valida nos metadados de segurança instanciados na base de dados, o perfil de acesso do usuário. Apenas os usuários com permissão de acesso conseguem se conectar a base de dados. Para cadatransação de confirmação (commit) solicitada, a implementação na camada Plugin é executado. Se o usuário solicitante tem permissão para as operações que ele / ela está tentando executar sobre os dados, a submissão é

executado. Caso contrário, ou seja, se o usuário não tem permissão para esta operação, a reversão é executada.

Os testes sistêmicos foram realizados com a utilização do Google Chrome versão 42.0.2311.135, divididos em três etapas. A primeira etapa tem o seu objetivo o contexto de autenticação (por exemplo, conectar-se aos SGBDG com o usuário mestre, criar uma nova conta e acessar o banco de dados com esse novo usuário; excluir o usuário e não pode acessar o banco com ele), os detalhes dos cenarios de testes desta estapa estão definidas no Apêndice 1. A segunda etapa tem o seu objetivo o contexto de permissão de DDL (por exemplo, a criação de uma estrutura de nó por um utilizador com criar permissão; criação de uma estrutura de nó por um utilizador sem criar permissão; exclusão de uma estrutura de nó por um utilizador com permissão cair) os detalhes dos cenarios de testes desta estapa estão definidas no Apêndice 2. A terceira etapa tem o seu objetivo o contexto de permissão de DML (por exemplo, a instanciação de um nó por um usuário com permissão de inserção; exclusão de um comentário de um usuário sem permissão de exclusão; manipulação de um nó (insert, update, delete) pelo proprietário do nó) os detalhes dos cenarios de testes desta estapa estão definidas no Apêndice 3.

5.2.1. Controle de Acesso

O controle de acesso do modelo de segurança para SGBDG foi desenvolvido como um *plugin* e colocado no diretório (*plugins*) do Neo4j. Após a inclusão do *plugin* com o modelo de segurança a URL de acesso ao SGBDG Neo4j passa a solicitar o usuário e senha para se conectar ao BDG.

Na criação do modelo de controle de acesso foi considerado o usuário "master" como o usuário administrador do SGBDG. Este usuário foi previamente configurado no *plugin* de controle de acesso com o *login* "master", senha "123" e deve ser o primeiro usuário a acessar o SGBDG, pois é o responsável pela criação de novos usuários e concessão de privilégios. A Figura 5.3, mostra a tela de autenticação do usuário com a solicitação do nome do usuário e senha.

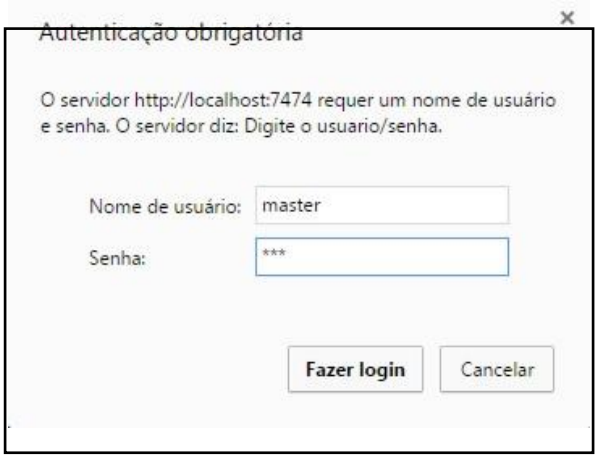

Figura 5.3: Tela de autenticação do usuário.

O usuário "master" deve cadastrar os demais usuários no próprio BDG, conforme a seção 4.1. Para isso deve-se utilizar a definição de metadados Meta-Usuario da Figura 4.1.

A Figura 5.4 apresenta um exemplo para criação de usuário com os seguintes atributos: Identificador cujo valor é "1", Nome cujo valor é "Usuario1", Data cujo valor é "30/10/2014" e o usuário que está criando o nó (proprietário do nó) cujo nome é "master".

```
create (u:USUARIO { Identificador: 1, Nome: "Usuario1", Senha: "123", Data: "30/10/2014", Proprietario: "master"});
```

Figura 5.4: Exemplo de criação do usuário.

Com exceção do usuário "master", o modelo de controle de acesso valida os demais usuários e senha no próprio BDG. Quando o usuário e senha digitados na tela de *login* não estão cadastrados no BDG, é apresentada uma mensagem informando que o usuário é inválido e solicitando um novo usuário para a conexão como mostra a Figura 5.5.

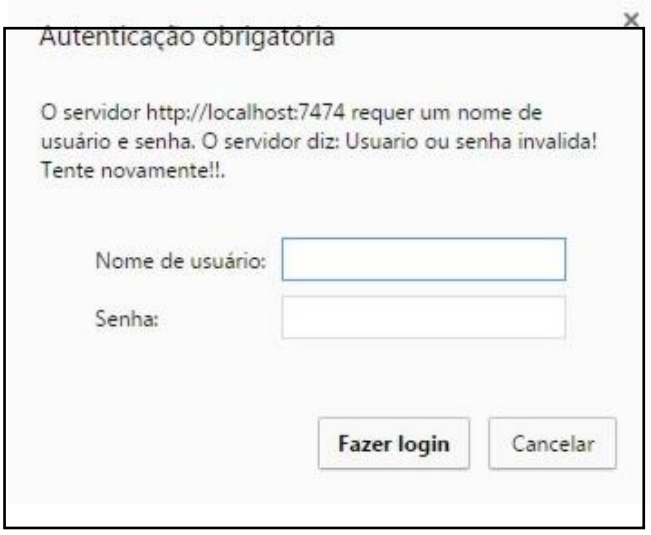

Figura 5.5: Tela com erro na autenticação do usuário.

Após a criação dos usuários, os mesmos foram relacionados com as permissões desejadas, conforme mostra as próximas seções deste capítulo.

Quando o usuário finaliza a sessão do Neo4j, é excluído o *cache* do *brouwser*, a fim de se evitar que o próximo usuário a logar no SGBDG herde alguma permissão do antigo usuário logado. Um dos maiores problemas enfrentados durante os testes da autenticação de usuário, estava acontecendo quando um usuário que tinha permissão saia do sistema e em seguida um usuário sem permissão autenticava e conseguia acesso ao BDG. As evidências de testes podem ser consultadas no Apêndice 1, este cenário específico se refere ao cenário de testes 004.

5.2.2. Privilégios Relacionados a Operações de DDL

Atualmente, o SGBDG Neo4j não possui definição de esquemas de nó ou relacionamento, mas, como o modelo de controle de acesso faz restrições de acesso a diferentes tipos de esquema de nó, propomos ao SGBDG Neo4j a definição de esquema de nó.

Para que um usuário tenha permissão de definir um esquema de nós a serem criados no grafo, o usuário deverá estar relacionado com o metadados de operação Meta-Operação,

instanciando as operações de DDL (CREATE, ALTER e DROP), conforme descrito na seção 4.2. Para isso deve-se utilizar a definição de metadados Meta-Operação da Figura 4.3.

A Figura 5.6, apresenta um exemplo de criação da operação, com os atributos Identificador cujo valor é 1 e o Nome cujo valor é “CREATE”.

```
create (op:OPERACAO {Identificador: 1, Nome: "CREATE"})
```

Figura 5.6: Exemplo de criação de operação.

É de responsabilidade do usuário “master” conceder permissões aos usuários. O usuário “master” também pode delegar esta permissão a outro usuário, para que o mesmo consiga fazer a concessão das permissões aos demais usuários.

Para conceder uma permissão ao usuário, o “master” deverá relacionar o nó de operação (Meta-Operação) de DDL ao usuário (Meta-Usuário), conforme descrito na seção 4.3.

A Figura 5.7, apresenta a associação entre o usuário de Identificador 1 com o privilégio de “CREATE”. Com esta associação o usuário passa a ter permissão de criar esquemas de nó no SGBDG.

```
// associação do usuário a permissão
MATCH (a:USUARIO),(b:OPERACAO)
where a.Identificador = 1 and b.Identificador = 1
create (a)-[r:PERMISSAO]->(b)
RETURN r
```

Figura 5.7: Exemplo de concessão de privilégios ao usuário.

A Figura 5.8 mostra a instância do exemplo anterior, onde o “Usuario1” possui permissão de criação de novos esquemas de nós. Após a concessão do privilégio, o usuário pode se conectar ao SGBDG e definir o esquema de nó.

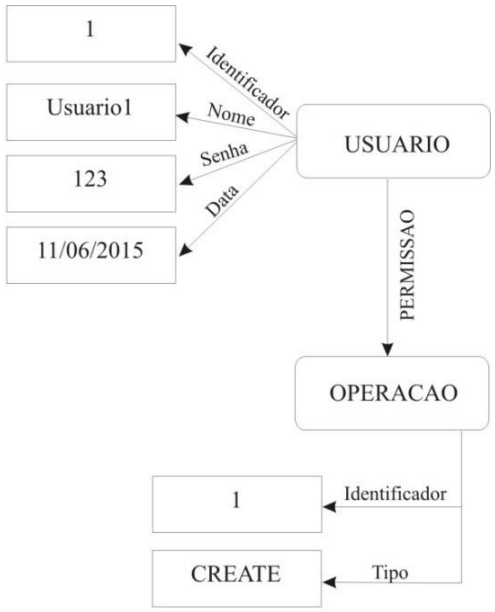

Figura 5.8: Instância do usuário com a permissão de CREATE.

A definição de um esquema possui a denominação do *label* do nó, que deve ser único no sistema. O usuário que está criando o nó passa a ser proprietário do nó e ter todo o tipo de permissão sobre ele, conforme seção 4.2. Os atributos do nó devem ter as informações do identificador do atributo, nome, tipo de dados e indicação de obrigatoriedade e estar relacionado ao nó.

A Figura 5.9, apresenta um exemplo de criação de esquema, onde ó *label* é "POST" e o proprietário é o "Usuario1".

```
//criação do esquema de nó
create (M:METANO { Label: "POST", Proprietario:"Usuario1"})
```

Figura 5.9: Exemplo de criação de esquema.

A Figura 5.10, apresenta um exemplo de criação de atributos associados ao esquema. Todo atributo deve ter um identificador, nome, tipo de dados e obrigatoriedade. Os atributos criados deverão ser associados a um esquema. Quando for instanciado um nó para este esquema, a instância do nó deve respeitar os atributos, com os respectivos tipos de dados e obrigatoriedade.

```
// atributos do esquema de nó
create (M:ATRIBUTO { Identificador: 1, Nome: "ID", Tipo: "NUMBER", Obrigatorio: "S"})
create (M:ATRIBUTO { Identificador: 2, Nome: "Titulo", Tipo: "STRING", Obrigatorio: "S" })
create (M:ATRIBUTO { Identificador: 3, Nome: "Reportagem", Tipo: "STRING", Obrigatorio: "S"})
```

Figura 5.10: Exemplo de criação dos atributos definidos para o esquema.

O resultado dos comandos da Figura 5.10, cria três atributos como segue:

- **ID**: Identificador com o valor "1", com tipo de dados numérico e obrigatório,
- **Título:** Identificador com valor "2", com tipo de dados caractere e obrigatório,
- **Reportagem:** Identificador com valor "3", com tipo de dados caractere e obrigatório.

A Figura 5.11, apresenta o exemplo de associação entre o esquema com seus respectivos atributos, como resultado o esquema "POST" está sendo associado aos atributos de Identificador 1, 2 e 3.

```
MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 1
CREATE (a)-[r:ATRIBUTO]->(b)
RETURN r

MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 2
CREATE (a)-[r:ATRIBUTO]->(b)
RETURN r

MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 3
CREATE (a)-[r:ATRIBUTO]->(b)
RETURN r
```

Figura 5.11: Exemplo da associação entre o esquema e seus respectivos atributos.

A Figura 5.12, mostra a instância do usuário "Usuario1" associado a permissão de "CREATE" com o METANO "POST" e seus atributos (ID, Titulo, Reportagem).

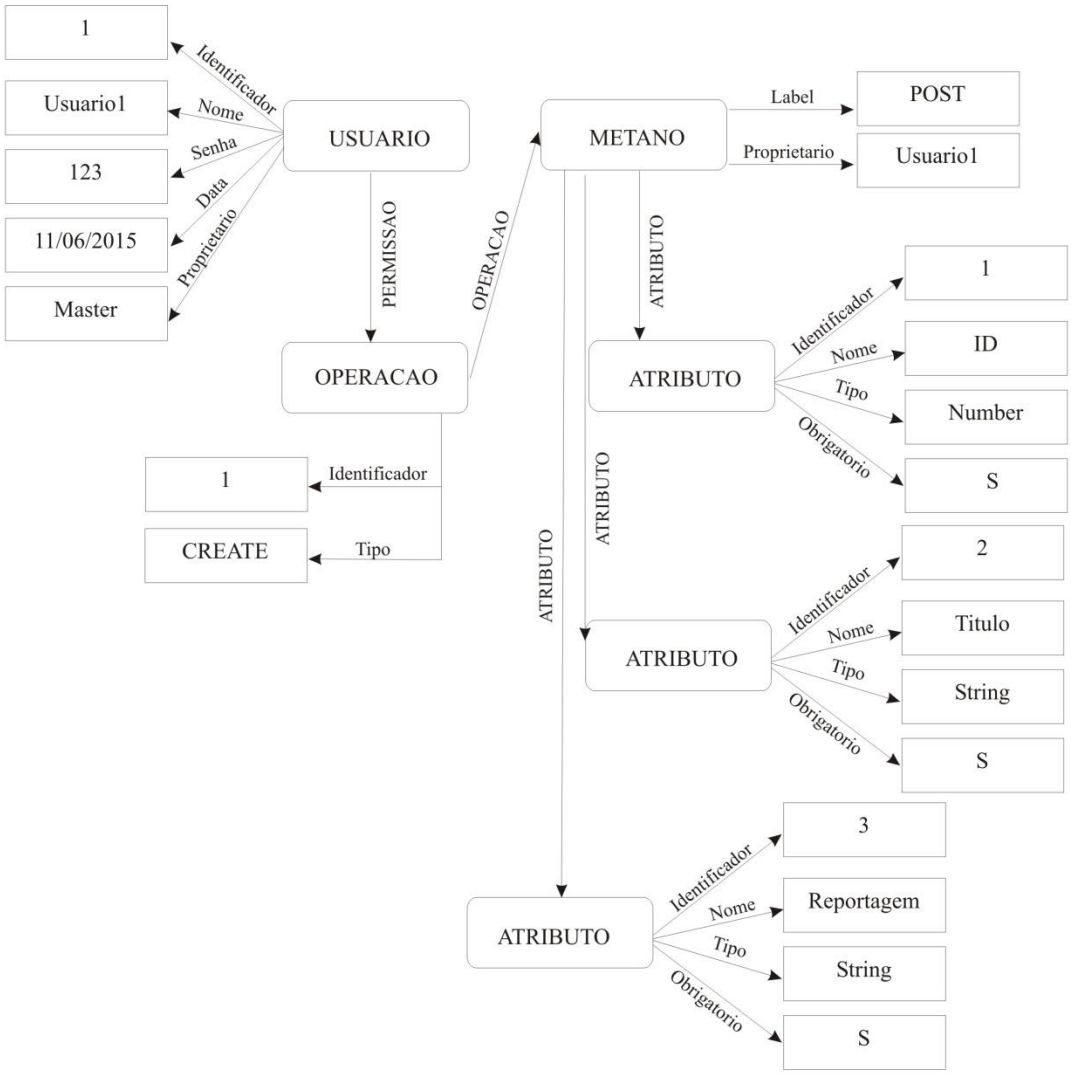

Figura 5.12: Instância do usuário com a permissão de CREATE e esquema de nó "POST".

Após criar um esquema de nó, o usuário que criou o esquema passa a ser seu proprietário e herda todas as permissões de DDL e DML para manipulação deste esquema.

Para alterar as informações dos atributos do esquema do nó o usuário deverá ter a permissão de ALTER. Os atributos não poderão ser alterados quando já existirem nós instanciados no BGD com o mesmo *label* do esquema.

A exclusão de esquemas depende da permissão de DROP associada. Apenas os esquemas que não tiverem nós instanciados podem ser excluídos.

5.2.3. Privilégios Relacionados a Operações de DML

Após criar o esquema de nó, o usuário poderá instanciar nós seguindo padrão definido no esquema. A concessão do privilégio para operação de DML, seção 4.4, está relacionada as operações INSERT, DELETE, UPDATE, SELECT e o esquema a que se refere. Por exemplo, se o usuário pode instanciar nós reportagens no grafo, ou seja, nós com *label* "POST", ele deve estar relacionado a operação de INSERT que está relacionada ao esquema "POST". Agora se o usuário pode apenas instanciar nós de comentários no grafo ele deve estar relacionado ao esquema "COMMENTS" e assim por diante.

Ao instanciar nós no BDG o usuário deverá respeitar os atributos de agora com a definição e obrigatoriedade definida. Para alterar as informações dos atributos de um nó instanciado o usuário deverá ter a permissão de UPDATE e a alteração deve respeitar a definição do esquema do nó. Por exemplo, não poderá ser removida uma informação do nó definida no esquema como um atributo obrigatório. A exclusão de nós instanciados depende da permissão de DELETE associada.

A Figura 5.13, mostra o usuário "Usuario1" com a permissão de DDL CREATE associada aos esquemas "POST" e "COMMENT" criados por ele. A Figura 5.13 também exemplifica algumas permissões de DML atribuídas ao usuário "Usuario2", para exemplificar a Figura 5.13 mostra a permissão para instanciar novas reportagens (POST) no BDG com a operação INSERT, e consultar as reportagens (POST) com a operação de SELECT associada ao esquema "POST". Na instanciação do nó com o *label* "POST" não há necessidade de colocar a da chave estrangeira (FK), conforme diagrama de entidade e relacionamentos da Figura 5.1 (FK_USER_ID), esta informação fica no relacionamento entre o nó do USUARIO e o POST.

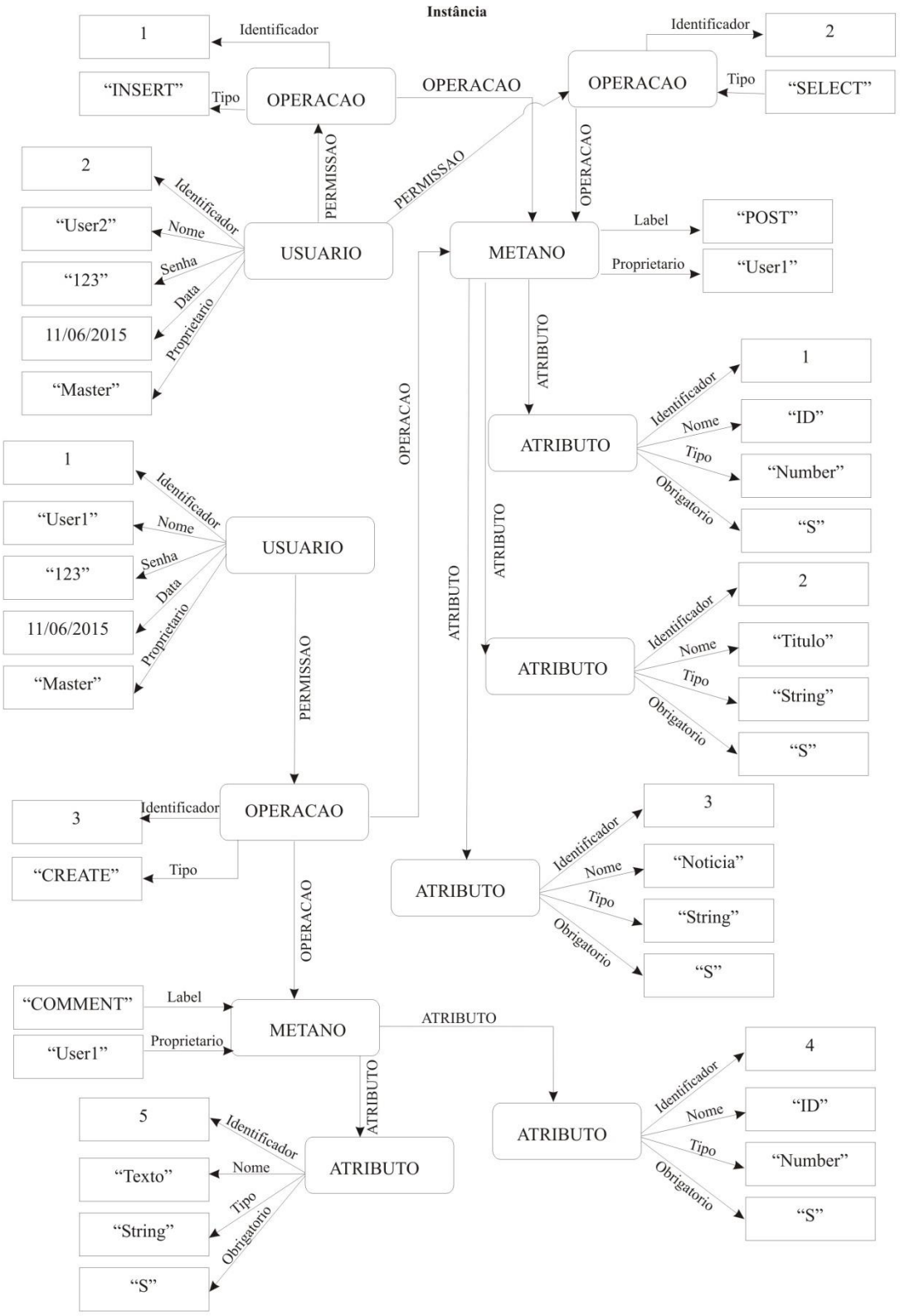

Figura 5.13: Instância do usuário com a permissão DDL e DML

5.3. Validação do Estudo de Caso

Após a configuração do *plugin* de controle de acesso no SGBDG Neo4j, conectamos ao SGBDG com o usuário "master". Este usuário é o responsável pela criação de todos os usuários com permissão para acesso ao SGDBG. Para o estudo de caso foram criados cinco usuários, com permissões distintas conforme apresenta a Tabela 5.1.

Tabela 5.1: Relação de permissões concedidas aos usuários.

Usuário	Label		Permissão						
	POST	COMMENTS	DDL			DML			
			C	A	D	I	U	D	S
master	X	X	X	X	X	X	X	X	X
Usuario1	X	X	X	X	X	X	X	X	X
Usuario2	X			X	X	X	X	X	X
Usuario3		X				X	X	X	X
Usuario4	X					X	X	X	X
		X				X	X	X	X
Usuario5		X				X			

Legenda:
Permissão DDL: C-Create; A – Alter; D – Drop
Permissão DML: I-Insert; U – Update; D – Delete; S - Select

O usuário "master" concedeu permissão ao "Usuario1" de criar, alterar e excluir esquemas de nó no BDG. Para isso associou a permissão CREATE, ALTER e DROP ao usuário. Ao usuário "Usuário2" concedeu a permissão para alterar e excluir esquemas, para isso associou a permissão de ALTER e DROP a esse usuário. Ao usuário "Usuário3" foi concedida a permissão para manipular nós do esquema COMMENTS, porém ele não tem permissão para incluir novas reportagens (POST). Já o usuário "Usuario4" tem permissões para comentar e incluir reportagens. Todos esses usuários tem direito total sobre todas as operações da DML. O usuário "Usuario5" é o único que tem restrições mais severas, podendo apenas incluir comentários, sem nenhum acesso aos comandos da DDL também.

Para validar a implementação utilizada como prova de conceito da abordagem foram criados diversos casos de testes com o objetivo de usar estas restrições impostas aos usuários criados. Os primeiros casos de testes executados envolveram os usuários “USUARIO1” e “USUARIO2”. Após criar os usuários e conceder a eles a permissão ambos conseguiam se conectar ao SGBDG Neo4j, porém quando o “Usuario2” acessava o BDG após o “Usuario1” ter se autenticado no ambiente, o “USUARIO2" herdava dele a permissão para criar esquemas de nó. Para solucionar o problema ao encerrar a sessão é excluído a *chache* do *browser*, fazendo com que ao se autenticar no SGBDG todas as permissões, do usuário que solicita a autenticação, são carregadas novamente de acordo com o modelo do controle de acesso estabelecido.

Outro problema na criação de esquemas é que o Neo4j não tem como característica nativa a definição de esquemas. Para poder criar o esquema necessário para o controle de acesso usamos os comandos para manipulação de nós do Cypher, tanto para definir o esquema da estrutura de acesso quanto para criar os atributos relacionados a esse esquema, como também, para associar esses atributos à estrutura criada.

Para relacionar o esquema aos atributos precisamos ter a permissão de selecionar o esquema e os atributos para então conseguir relacionar. Como o usuário com a permissão de criação de esquemas de nó, no início não teria permissão de consulta (SELECT), não era possível fazer este relacionamento. Para solucionar o problema definimos que todo usuário que cria um esquema passa a ser proprietário do esquema e recebe todas as permissões sobre ele, ALTER, DROP, INSERT, UPDATE, DELETE e SELECT.

Após os ajustes nos conectamos na base de dados com o usuário “Usuario1”, e criamos os esquemas de nós (POST e COMMENTS) para implantar o modelo de controle de acesso a estes esquemas e realizar os testes. O usuário “Usuario1” recebeu automaticamente todas as permissões aos dois esquemas criados. Ao se conectar como “Usuario2” já não havia problema de acesso não autorizado, uma vez que a limpeza da *cache do brouwser* solucionou a questão.

Os testes com o “Usuario3” que tinha permissão para manipular nós do esquema COMMENTS, trouxe problemas para distinguir quando se trabalhava com o esquema ou com a instanciação. No momento de se instanciar um nó deve-ser validar os atributos e só finalizar a operação quando todos os atributos estiverem de acordo com a estrutura definida no esquema. Como para criar um esquema e para instanciar um nó do esquema usamos o mesmo comando do Cypher (create), não era possível distinguir se a operação era de definição de um novo esquema

ou instanciação de um nó. Para solucionar esse problema, lançamos mão do *label*, usado no comando. Quando existir um esquema para o *label*, e o usuário tiver permissão de INSERT neste esquema, os atributos no esquema são validados e somente se estiver tudo correto a operação é efetivada. Após os ajustes, foi validada a operação de INSERT do *label* COMMENTS sob o comando do "Usuario3", mostrando o sucesso da solução.

Quando o *label* usado no create for METANO, é feita a validação da permissão a "CREATE" do usuário em questão e se o esquema que se deseja criar não é um esquema já existente no BDG.

Para alterar os atributos que definem um esquema de nó, o usuário deve ter a permissão de "ALTER" e que não haja nós instanciados para este esquema. A única operação de alteração permitida quando existem nós instanciados é a de tornar um atributo opcional.

Não é possível excluir nós que tiverem relacionamentos com outros nós no Neo4j, então para excluirmos um esquema primeiramente temos que excluir o relacionamento do esquema, para em seguida excluir o esquema e os atributos. Para a exclusão deste tipo de relacionamento (esquema x atributos) é validado se existe a permissão a "DROP" do usuário e se não existem nós instanciados com este esquema, afim de evitar que fiquem nós sem esquema na base de dados. Outros tipos de relacionamentos não estão sendo validados neste modelo de controle de acesso.

A exclusão de informações nas instancias da base só é possível se o usuário tem a permissão de "DELETE".

O usuário só tem permissão de consulta aos nós se for o proprietário do esquema daquele nó ou se tiver permissão de consulta (SELECT) para o mesmo. Essa restrição foi validada utilizando-se o "Usuario5". Como o "Usuário5" tem apenas permissão de INSERT mas não tem permissão de consulta e nem é proprietário do esquema COMMENTS, não consegue consultar seus próprios nós.

Todas essas restrições foram validadas utilizando-se casos de teste apropriados. Alguns cenários de testes estão detalhados nos Apêndices 1. 2 e 3.

5.4. Considerações finais

Este capítulo apresentou o estudo de caso do modelo de controle de acesso para o SGBDG que foi implementado no SGBDG Neo4j, aplicado a um *website* de notícias.

Com o usuário DBA, criamos diversos usuários no BDG. Ao conceder o privilégio de CREATE ao usuário "Usuário1", o DBA passou a ter permissão de criar no BDG esquemas de nós (Meta-Nó). E com ele criamos dois esquemas POST e COMMENTs. A cada esquema de nó criado foi definido quais os atributos (Meta-Atributo) este esquemas deveriam ter. O usuário "Usuario1" passou a ser o proprietário dos esquemas e ter todos os privilégios sobre ele.

Após a criação dos esquemas o usuário "master" concedeu diferentes tipos de permissões a eles. Por exemplo, o usuário "Usuário5" apenas poderia comentar as reportagens, enquanto o usuário "Usuario4" poderia comentar e publicar novas reportagens. Todos os testes foram realizados com sucesso.

Com o estudo de caso, mostramos que é possível criar um controle de acesso aos SGBDGs com uma menor granularidade e definição de papéis. Além disso, a criação de esquemas ao SGBDG trouxe uma maior garantia de integridade dos dados.

Toda a implementação foi desenvolvida como *plugin* utilizando a linguagem Java e configurada no diretório de *plugins* do Neo4j.

6. Conclusão

Os SGBDs relacionais não são mais tão eficientes para as aplicações complexas, com grandes volumes de dados, semi-estruturados ou não estruturados que requerem alta disponibilidade e escalabilidade. Os SGBDs não somente relacionais são uma opção para este tipo de problema, mas precisam melhorar a parte da segurança, como autenticação de usuários, controle de acesso, criptografia, entre outros

Esse trabalho apresentou, com o auxílio de metadados, um modelo de controle de acesso para garantir que usuários não autorizados, não possam ter acesso ou modificar os dados armazenados em um SGBDG.

Foram criados cinco estruturas de metadados para o controle de acesso ao SGBDG:

- Meta-Usuário: usuários que possui permissão de acesso ao BDG.
- Meta-Grupo: agrupa os usuários em grupos com as mesmas necessidades e mesmos tipos de permissões.
- Meta-Operação: todos os privilégios do BDG, create, drop, alter, insert, delete, update ou select.
- Meta-Nó: nome e proprietário da estrutura de nó.
- Meta-Atributo: atributos que pertencem à estrutura de nó.

Os metadados são relacionados de acordo com as permissões concedidas para os usuários. Um usuário (Meta-Usuário) pode receber privilégios individualmente ou herdar esses privilégios de um grupo de usuários (Meta-Grupo). Os privilégios do BDG estão relacionados as operações (Meta-Operação) que os usuários podem realizar no BDG.

As permissões de DDL (CREATE, ALTER, DROP) permite ao usuário criar, alterar ou excluir esquemas de nós. A cada esquema de nó (Meta-Nó) criado, é definido quais os atributos (Meta-Atributo) este esquema de nó deve ter, ou seja, quais as propriedades, ou atributos que cada instância do esquema de nó deve ter.

Para que o usuário tenha permissão de manipular a instância do BDG o usuário deve receber as permissões de DML (INSERT, DELETE, UPDATE, SELECT). Ao instanciar um nó, as propriedades deste nó devem respeitar os atributos definidos no esquema.

O modelo de controle de acesso criado foi implementado em um estudo de caso como um *plugin* para o SGBDG Neo4j e aplicado a um *website* de notícias.

Com a utilização do modelo de controle de acesso foi possível criar um administrador ao SGBDG Neo4j e conceder diferentes tipos de permissões aos usuários. Com isto, podemos restringir o acesso do usuário de acordo com o seu perfil de acesso.

Tendo em vista os bons resultados para as restrições no controle de acesso aos nós propõe-se, como trabalhos futuros, que seja estendido o controle de acesso ao nível de relacionamentos, restringindo o acesso dos usuários de acordo com as propriedades dos relacionamentos além dos nós. Outra melhoria futura poderia ser a inclusão da criptografia no armazenamento de usuários e senhas para aumentar a segurança dos dados

A criação de comandos específicos para DDL facilitaria a inclusão de esquemas na base de dados, já que hoje, primeiramente temos que definir os esquemas, em seguida os atributos e só depois associar os esquemas aos seus respectivos atributos. Esta também seria uma importante contribuição como trabalhos futuros.

7. Referências

ANGLES, R.: "*A Comparison of Current Graph Database Models"*. In: Proc. of IEEE 28th International Conference on Data Engineering Workshops (ICDEW), pp. 171 – 177, 2012. .

ANGLES, R.; GUTIERREZ, C.: *"Survey of graph database models".* ACM Comput. Surv. 40, 1, Article 1, 2008.

ACCUMULO: "The Apache Accumulo™ sorted, distributed key/value store is a robust, scalable, high performance data storage and retrieval system". Available at http://accumulo.apache.org/. Last access on march/2016.

AZURE: Windows Azure Table Storage Service. Disponível em: http://www.windowsazure.com/en-us/develop/net/how-to-guides/table-services/#concepts. Ultimo Acesso: Janeiro/2015.

BOLLOBÁS, B.: "*Modern Graph Theory*". Graduate Texts in Mathematics – GTM, Vol. 184, Editora Springer, 1998. pp.394.

BONDY J.A.; MURTY U.S.R.: *"Graph Theory"* Graduate Texts in Mathematics – GTM, Editora: Springer, 2008.

CALLIL, A.; MELLO R. S.: "*SimpleSQL: A Relational Layer for SimpleDB*". Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brazil, Springer-Verlag Berlin Heidelberg 2011.

CASTELLTORD, A., LAURENT, A.: "Representing History in Graph-Oriented NoSQL DataBases: A Versioning System" Digital Information Management (ICDIM), 2013 Eighth International Conference on. Page(s) 228-234.

CATELL, R.: *"Scalable SQL and NoSQL data stores".* ACM SIGMOD Record , Volume 39 Issue 4, 2010.

CERQUEIRA, A.: "*Introdução a Metadados*". Disponível em http://www.linhadecodigo.com.br/artigo/298/introducao-a-metadados.aspx Ultimo acesso em Novembro/2014.

COLOMBO, P., FERRARI E.: "Representing Enhancing MongoDB with Purpose based Access Control" in IEEE Transactions on Dependable and Secure Computing, vol. PP, no.99, pp.1-8, 2015. DIESTEL, R.: "*Graph Theory*" Graduate Texts in Mathematics - – GTM, Editora: Springer, 3ª Edição, 2005. pp. 417.

DoD "*Department of Defense Trusted Computer System Evalution Criteria*"., 1985 Disponível em: http://csrc.nist.gov/publications/history/dod85.pdf . Ultimo acesso em Janeiro/2015.

DYNAMODB: Disponível em: http://aws.amazon.com/pt/dynamodb/. Ultimo Acesso Outubro/2015.

ELMASRI, R., NAVATHE, P.: "Sistemas de Banco de Dados" 4ª Edição.Editora. Pearson.

HBASE: Disponível em: http://hbase.apache.org/0.94/book/hbase.accesscontrol.configuration.html. Ultimo acesso Dezembro/2015.

HAN, J.; HAIHONG, E.; GUAN LE; JIAN DU: "*Survey on NoSQL database*". In: Proc. Of IEEE 6th International Conference on Pervasive Computing and Applications (ICPCA), pp. 363 – 366, 2011.

KULKARNI D.: "*A fine-grained access control model for key-value systems*". In Proceedings of the third ACM conference on Data and application security and privacy (CODASPY), 2013, pp. 161–164.

LEAVITT, N.; "*Will NoSQL Databases Live Up to Their Promise?*", IEEE Computer ; v.43 n.2, pp: 12-14, 2010.

LÓSCIO, F. B.; OLIVEIRA, H. R.; PONTES, J. C. S.: *"NoSQL no desenvolvimento de aplicações Web colaborativas".* VIII Simpósio Brasileiro de Sistemas Colaborativos, 2011. Disponível em: http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf. Ultimo acesso em Dezembro/2015.

MARTINO, L.D.; NI Q; LIN D.; BERTINO, E.: "*Privacy-Aware Role-Based Access Control*", IEEE Security Privacy, vol. 7, no. 4, p. 35-43, ago. 2009.

MONGODB: Disponível em: http://www.mongodb.org/. Último acesso em: Janeiro/2013.

NEO4J: Disponível em http://neo4j.org - Ultimo acesso em Janeiro/2016.

NEWMAN, M.E.J.: "*The Structure and Function of Complex Networks.*" SIAM Review. Vol. 45(2): 167-256. 2003.

OKMAN, L.; GAL-OZ, N.; GONEN, Y.; GUDES, E.; ABRAMOV, J.: *"Security Issues in NoSQL Databases."* In: Proc. of IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 541-547, 2011.

PEREIRA, V.; “Redes complexas em presença de falhas induzidas”. Disponível em: http://www.bibliotecadigital.unicamp.br/document/?code=000783381, 2010. Último acesso em Outubro de 2012.

REDIS: Disponível em http://redis.io/. Último acesso em Janeiro/2013.

SANDHU, R. S.: “*Role-based Access Control*”, in Advances in Computers, vol. Volume 46, Elsevier, 1998, p. 237-286.

SRINIVAS S.; NAIR A. *“Security maturity in NoSQL databases are they secure enough to haul the modern IT applications?”*. International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2015, pp.739-744

STONEBRAKER, M.; *“SQL Databases v. NoSQL Databases”.* Communications of the ACM, v.53 n.4, 2010.

TOKIO CABINET: Disponível em: http://fallabs.com/tokyocabinet/. Último acesso em Janeiro de 2013.

VICKNAIR C; MACIAS M, ZHAO Z, NAN X, CHEN Y, WILKINS D. “*A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective,* ACM SE’10, 2010.

WILSON, J. R.: “*Introduction to Graph Theory*”. 4ª Edição. Editora Prentice Hall , 1996.

YANG, L.: “*Teaching database security and auditing*”. ACM SIGCSE '09, v.1, issue 1, pp. 241-245, 2009.

YUBIN G. ; LIANKUAN Z. ; LIANKUAN, L. F. e XIMING L.: “*A solution for privacy preserving data manipulation and query on NoSQL database*”. Journal of Computers, vol. 8, Issue 6, 2013, pp. 1427–1432.

YELLOWFIN. Disponível em http://www.yellowfin.com.br/apresentacao. Ultimo acesso em Novembro/2012.

ZAHID, A.; MASOOD, R. SHIBLI, M.A.; *“Security of Sharded NoSQL Database: A Comparative Analysis”* . Conference on Information Assurance and Cyber Security. IEEE. 2014.

ZHU H. ; K. LÜ, R. JIN: “A *practical mandatory access control model for XML databases, Information Sciences*”, vol. 179, Issue 8, 2009, pp. 1116-1133.

APENDICE 1

Evidências de testes do Controle de Acesso

Identificador do Caso de Teste: 001
Objetivo do teste: Se conectar ao banco de dados com o administrador do SGBDG “master”
Resultado esperado: Acessar o banco de dados neo4j
Data da Execução: 12/01/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Acessar o Neo4j (HTTP://localhost:7474/) 2. Usuário: master / senha 123
Histórico dos testes: 1. SUCESSO: Conexão realizada com sucesso.

Identificador do Caso de Teste: 002
Objetivo do teste: Criar uma nova conta e acessar o banco com o novo usuário.
Resultado esperado: Incluir um novo usuário e conseguir acesso ao banco com o novo usuário.
Data da Execução: 12/05/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Logar com o usuário “master” 2. Incluir um novo usuário Comando: 3. create (u:USUARIO { Identificador: 1, Nome: “Usuario1”, Senha: “123”, Data: “30/10/2014”, Proprietario: “master”}); Fechar a janela 4. Logar novamente com o novo usuário (“Usuario1” / 123).
Histórico dos testes: 1. SUCESSO: Conexão realizada com sucesso.

Identificador do Caso de Teste: 003
Objetivo do teste: Informar um usuário inválido.
Resultado esperado: Não conseguir acessar o banco de dados Neo4j
Data da Execução: 12/01/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes:

1. Logar no SGBDG Neo4j com um usuário que não está cadastrado no banco de dados. Exemplo: usuário: teste / senha: 123 2. Logar com um usuário cadastrado no banco de dados mas digitar um senha inválida. Exemplo: usuário: usuario1 / senha: 444
Histórico dos testes: 1. SUCESSO: Mensagem de erro informando que o usuário não poderia se conectar ao SGBDG Neo4j.

Identificador do Caso de Teste: 004
Objetivo do teste: Deslogar de um usuário valido e logar com um inválido.
Resultado esperado: Não ter permissão de acesso ao Neo4j
Data da Execução: 12/05/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Logar com o usuário: "Usuario1" / senha: "123" 2. Fechar a janela 3. Logar com um usuário inválido: "teste" / senha: "123"
Histórico dos testes 1. ERRO: ao deslogar com um usuário válido, o próximo usuário inválido se conectava normalmente. 2. CORREÇÃO: limpar o cachê ao se deslogar com usuário. 3. SUCESSO: não foi possível fazer o login com o usuário inválido.

Identificador do Caso de Teste: 005
Objetivo do teste: Excluir usuário
Resultado esperado: Excluir usuário e não conseguir mais logar no banco com ele.
Data da Execução: 12/05/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Logar com o usuário: master / senha: 123 2. Excluir usuário. Comando: MATCH (n:USUARIO) where n. Identificador = 5 DELETE n; 3. Fechar a janela 4. Logar com o usuário: "Usuário5" / senha: 123
Histórico dos testes: 1. SUCESSO: Após a exclusão do usuário não foi mais possível se conectar com o usuário excluído no SGBDG Neo4j.

Identificador do Caso de Teste: 006
Objetivo do teste: Incluir novos usuários
Resultado esperado: Cadastrar novos usuários e conseguir logar com todos eles.
Data da Execução: 12/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Logar com o usuário: master / senha: 123 2. Criar novos usuários Comandos: create (u:USUARIO { Identificador: 2, Nome: "Usuario2", Senha: "123", Data: "30/10/2014", Proprietario: "master"}); create (u:USUARIO { Identificador: 3, Nome: "Usuario3", Senha: "123", Data: "30/10/2014", Proprietario: "master"}); create (u:USUARIO { Identificador: 4, Nome: "Usuario4", Senha: "123", Data: "30/10/2014", Proprietario: "master"}); create (u:USUARIO { Identificador: 5, Nome: "Usuario5", Senha: "123", Data: "30/10/2014", Proprietario: "master"}); 3. Logar com cada um deles e verificar se está correto.
Histórico dos testes: 1. SUCESSO: Todos os usuários cadastrados conseguiram se conectar ao SGBDG.

APENDICE 2

Evidências de testes referente aos privilégios relacionados a operações de DDL

Identificador do Caso de Teste: 001

Objetivo do teste: Criar um novo esquema de nó com um usuário que possua privilégio de acesso

Resultado esperado: Conseguir criar um novo esquema de nó.

Data da Execução: 19/10/2015

Resultado: ( ) Sucesso / (X) Falha

Detalhes do testes:

1. Logar com o usuário: master / senha: 123
2. Criar a permissão CREATE concede-la ao usuário: “Usuario1” / senha: “123”.

Comando:

```
-- Criação da permissão CREATE
create (op:OPERACAO {Identificador: 1, Nome: "CREATE"});

-- Relacionamento USUARIO x OPERAÇÃO
MATCH (a:USUARIO),(b:OPERACAO)
where a.Identificador = 1 and b.Identificador = 1
create (a)-[r:PERMISSAO]->(b)
RETURN r
```

3. Fechar a janela
4. Logar com o usuário1 / senha: 123
5. Criar o novo esquema de nó, “POST”

Comando:

```
create (M:METANO { Label: "POST", Proprietario:"Usuario1"});

create (M:ATRIBUTO { Identificador: 1, Nome: "ID", Tipo: "NUMBER", Obrigatorio: "S",
Proprietario:"Usuario1"})
create (M:ATRIBUTO { Identificador: 2, Nome: "Titulo", Tipo: "STRING", Obrigatorio:
"S", Proprietario:"Usuario1" })
create (M:ATRIBUTO { Identificador: 3, Nome: "Reportagem", Tipo: "STRING",
Obrigatorio: "S", Proprietario:"Usuario1"})
create (M:ATRIBUTO { Identificador: 4, Nome: "UserId", Tipo: "STRING", Obrigatorio:
"S", Proprietario:"Usuario1"})

// relacionamento USER x ATRIBUTO
MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 1
CREATE (a)-[r:ATRIBUTO]->(b)
```

RETURN r

MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 2
CREATE (a)-[r:ATRIBUTO]->(b)
RETURN r

MATCH (a:METANO),(b:ATRIBUTO)
WHERE a.Label = "POST" AND b.Identificador = 3
CREATE (a)-[r:ATRIBUTO]->(b)
RETURN r

Histórico dos testes

1. **ERRO:** Não foi possível criar o esquema de nó, erro intermitente: Neo.DatabaseError.Statement.ExecutionFailure**.**
2. **CORREÇÃO:** Com o Neo4j, o ponto de entrada para trabalhar com transações é a classe EmbeddedGraphDatabase, através do seu método beginTx(), definido na interface GraphDatabaseService, que retorna um objeto do tipo Transaction. A partir do momento que o beginTx() é invocado, todas as operações envolvendo o banco de dados, naquela Thread, é considerado parte da transação, até o momento que o métidos sucess() ou failure() sejam invocados na mesma. Por fim, a transação deve ser finalizada através do método finish() só após a conclusão de todas as validações.
3. **SUCESSO:** ao se conectar com o usuário "Usuario1", foi possível criar o esquema POST.

Identificador do Caso de Teste: 002

Objetivo do teste: Criar um novo esquema de nó com um usuário que não possua privilégio de acesso

Resultado esperado: Não conseguir criar o esquema de nó

Data da Execução: 19/10/2015

Resultado: (X) Sucesso / ( ) Falha

Detalhes do testes:

1. Logar com o usuário: master / senha: 123
2. Logar com o novo usuário: "usuario2" / senha: 123
3. Criar o novo esquema de nó, "USER"

Comando;

create (M:METANO { Label: "COMMENTS", Proprietario:"Usuario2"});

Histórico dos testes:

1. SUCESSO: Não foi possível criar esquemas de nós para usuários de não tinham permissão de CREATE.

Identificador do Caso de Teste: 003
Objetivo do teste: Alterar um esquema de nó, com usuário que tenha permissão de alteração de esquemas de nós. Não existe neste momento nó instanciados na base para o esquema.
Resultado esperado: Conseguir alterar o esquema de nó
Data da Execução: 19/10/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 1. Logar com o usuário "Usuario1" e alterar o atributo identificador do usuario como opcional no esquema POST. MATCH (a:METANO)-[r:ATRIBUTO]-(x) WHERE a.Label = "POST" and x.Identificador = 4 SET x.Obrigatorio = 'N'
Histórico dos testes: 1. SUCESSO: Foi possível realizar a alteração.

Identificador do Caso de Teste: 004
Objetivo do teste: Alterar um esquema de nó, com usuário que não tenha permissão de alteração de esquemas de nós.
Resultado esperado: Não conseguir criar o esquema de nó
Data da Execução: 19/10/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: 2. Logar com o usuário "Usuario2" e alterar o atributo identificador do usuario como opcional no esquema POST. MATCH (a:METANO)-[r:ATRIBUTO]-(x) WHERE a.Label = "POST" and x.Identificador = 4 SET x.Obrigatorio = 'N'
Histórico dos testes: 1. SUCESSO: Não foi possível realizar a alteração.

Identificador do Caso de Teste: 005
Objetivo do teste: Excluir um esquema de nó, com um usuário que possua permissão de DROP e sem nós instanciados para este esquema
Resultado esperado: Conseguir excluir o esquema de nó e seus atributos
Data da Execução: 09/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: Excluir o esquema "POST"

Comando:

```
-- Excluir o relacionamento
MATCH (a:METANO)-[x:ATRIBUTO]-(n)
WHERE a.Label = "POST"
DELETE x;
-- Excluir metano
MATCH (a:METANO) WHERE a.Label = "POST" DELETE a;
-- Excluir
MATCH (a:ATRIBUTO) WHERE a.Identificador = 1 DELETE a;
MATCH (a:ATRIBUTO) WHERE a.Identificador = 2 DELETE a;
MATCH (a:ATRIBUTO) WHERE a.Identificador = 3 DELETE a;
```

Histórico dos testes

1. SUCESSO: Foi possível excluir o nos com os atributos.

Identificador do Caso de Teste: 006
Objetivo do teste: Excluir um esquema de nó, com um usuário que NÃO possua permissão de DROP e sem nós instanciados para este esquema
Resultado esperado: NÃO conseguir excluir o esquema de nó e seus atributos
Data da Execução: 09/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: Comando: -- Excluir o relacionamento MATCH (a:METANO)-[x:ATRIBUTO]-(n) WHERE a.Label = "POST" DELETE x;
Histórico dos testes SUCESSO: Não foi possível excluir o relacionamento entre os nós e os atributos. Com os relacionamentos não é permitido também excluir os nós e atributos.

Identificador do Caso de Teste: 007
Objetivo do teste: Excluir um esquema de nó, com um usuário que possua permissão de DROP e com nós instanciados para este esquema
Resultado esperado: Não conseguir excluir o esquema de nó e seus atributos
Data da Execução: 09/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: Excluir o esquema “POST” Comando: -- Excluir o relacionamento MATCH (a:METANO)-[x:ATRIBUTO]-(n)

WHERE a.Label = "POST" DELETE x; -- Excluir metano MATCH (a:METANO) WHERE a.Label = "POST" DELETE a; -- Excluir MATCH (a:ATRIBUTO) WHERE a.Identificador = 1 DELETE a; MATCH (a:ATRIBUTO) WHERE a.Identificador = 2 DELETE a; MATCH (a:ATRIBUTO) WHERE a.Identificador = 3 DELETE a;
Histórico dos testes 1. SUCESSO: Não foi possível excluir o esquema

Identificador do Caso de Teste: 008
Objetivo do teste: Alterar um esquema de nó, com um usuário que possua permissão de ALTER e com nós instanciados para este esquema
Resultado esperado: Não conseguir alterar o esquema de nó e seus atributos
Data da Execução: 09/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: Excluir o esquema “´POST” Comando: -- Excluir o relacionamento MATCH (a:METANO)-[x:ATRIBUTO]-(n) WHERE a.Label = "POST" DELETE x; -- Excluir metano MATCH (a:METANO) WHERE a.Label = "POST" DELETE a; -- Excluir MATCH (a:ATRIBUTO) WHERE a.Identificador = 1 DELETE a; MATCH (a:ATRIBUTO) WHERE a.Identificador = 2 DELETE a; MATCH (a:ATRIBUTO) WHERE a.Identificador = 3 DELETE a;
Histórico dos testes 1. SUCESSO: Não foi possível alterar o esquema.

Identificador do Caso de Teste: 009
Objetivo do teste: Repetir todos os testes mas com a permissão concedida ao grupo e o grupo associado ao usuário.
Resultado esperado: Deve ser transparente a aplicação que a permissão vem do grupo. As permissões devem funcionar da mesma forma de quando é concedida diretamente ao funcionário,
Data da Execução: 28/12/2015
Resultado: (X) Sucesso / ( ) Falha

Histórico dos testes
1. SUCESSO: Todos os testes realizados anteriormente tiveram os mesmos resultados com a permissão concedida ao grupo.

Identificador do Caso de Teste: 006
Objetivo do teste: Incluir um esquema com o mesmo nome de um esquema existente.
Resultado esperado: Não ser possível cadastrar esquemas com o mesmo nome de esquemas já incluídos.
Data da Execução: 12/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhes do testes: create (M:METANO { Label: "POST", Proprietario:"Usuario1"});
Histórico dos testes: 2. SUCESSO: Não foi possível incluir esquemas com o mesmo nome.

APENDICE 3

Evidências de testes referente aos privilégios relacionados a operações de DML

Identificador do Caso de Teste: 001
Objetivo do teste: Instanciar um nó de um esquema cujo usuário tem permissão de INSERT ao esquema.
Resultado esperado: Permitir instanciar nós com o label que possui permissão.
Data da Execução: 28/11/2015
Resultado: (X) Sucesso / ( ) Falha

Detalhe do teste:
// criar a permissão INSERT
create (op0:OPERACAO {Identificador: 10, Nome: "INSERT"});
// associar ao usuario3
MATCH (a:USUARIO),(b:OPERACAO)
WHERE a.Identificador = 3 AND b.Identificador = 10
CREATE (a)-[r:PERMISSAO_INSERT]->(b)
RETURN r
// Associar a operação de insert ao esquema POST
MATCH (a:OPERACAO),(b:METANO)
WHERE a.Identificador = 10 AND b.Label = "POST"
CREATE (a)-[r:OPERACAO]->(b)
RETURN r
// Se conectar com o usuario 3
create (n:POST {ID: 1, Titulo: "Chuva", Reportagem: "As chuvas fazem subir os reservatorios "});

Histórico dos testes 1. SUCESSO: Foi possível criar instancias com o label POST.

Identificador do Caso de Teste: 002
Objetivo do teste: Instanciar um nó sem respeitar as regras definidas ao nó no esquema.
Resultado esperado: Não permitir instanciar nós quando não for respeitada as regras do esquema..
Data da Execução: 28/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe do teste: // Não incluir o Título na instancia no POST create (n:POST {ID: 1,Reportagem: "As chuvas fazem subir os reservatórios "});

Histórico dos testes 1. SUCESSO: Não foi possível criar instancias com o label POST.

Identificador do Caso de Teste: 003
Objetivo do teste: Tentar instanciar um nó sem ter permissão de INSERT ao esquema
Resultado esperado: Nao permitir instanciar nós quando não possui permissão aquele esquema de nós.
Data da Execução: 28/11/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe do teste: // Se conectar com um usuário que não tenha permissão de INSERT create (n:POST {ID: 2, Titulo: "Teste", Reportagem: "Teste"});
Histórico dos testes 1. SUCESSO: Não foi possível criar instancias sem permissão de INSERT.

Identificador do Caso de Teste: 004
Objetivo do teste: Tentar instanciar um nó que não tenha esquema definido.
Resultado esperado: Não permitir instanciar nós sem o esquema definido e sem ter permissão ao esquema.
Data da Execução: 01/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe do teste: // Se conectar com qualquer usuário que tenha permissão de insert create (n:COMENTARIO {ID: 2, Titulo: "Teste", Reportagem: "Teste"});
Histórico dos testes 1. SUCESSO: Não foi possível criar instancias sem esquema definido.

Identificador do Caso de Teste: 005
Objetivo do teste: Manipular um nó (inserir, excluir, alterar) com o usuário proprietários do esquema do nó.
Resultado esperado: Permitir instanciar/alterar/ excluir nós sendo o proprietário do nó
Data da Execução: 01/12/2015
Resultado: (X) Sucesso / ( ) Falha

Detalhe do teste:
// Se conectar com o usuário proprietário do esquema
create (n:POST {ID: 2, Titulo: "Teste 2", Reportagem: Teste 2})
 	
// Alterar
MATCH (n:POST) where n.ID =1 SET n.Titulo = 'Chuvas no Brasil'

// Excluir MATCH (n:POST) where n.ID =1 delete n
Histórico dos testes 1. ERRO: Somenta a operação de inclusão e alteração do nó deram certo; 2. CORRECAO: Não estava retornando nas permissões a permissão de DELETE 3. SUCESSO: todas as operações deram certo.

Identificador do Caso de Teste: 006
Objetivo do teste: Conceder permissão de UPDATE ao esquema POST e alterar os comentários criados no sistema.
Resultado esperado: Permitir alterar comentários no sistema
Data da Execução: 02/12/2015
Resultado: (X) Sucesso / ( ) Falha

Detalhe dos testes
// criar permissão
create (op0:OPERACAO {Identificador: 12, Nome: "UPDATE"});
// Associar a permissão ao Usuario3
MATCH (a:USUARIO),(b:OPERACAO)
WHERE a.Identificador = 3 AND b.Identificador = 12
CREATE (a)-[r:PERMISSAO_INSERT]->(b)
RETURN
// Alterar valor da propriedade Título
MATCH (n:POST) where n.ID =1 SET n.Titulo = 'Chuvas no Brasil'

Histórico dos testes 2. SUCESSO: Foi possível altear o nó.

Identificador do Caso de Teste: 007
Objetivo do teste: Conceder permissão de UPDATE ao esquema POST e alterar o nó removendo uma propriedade obrigatória.
Resultado esperado: Não permitir alterar nós de modo que não respeite as regras do esquema.
Data da Execução: 20/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe dos testes: // Alterar um atributo obrigatório para opcional MATCH (n:POST) where n.ID =1 SET n.Titulo = ''
Histórico dos testes 1. ERRO: Foi possível remover o valor da propriedade mesmo ela sendo obrigatória na definição do esquema. 2. CORREÇÃO:Validar os campos obrigatórios na alteração da instancia. 3. SUCESSO: Não foi possível alterar um atributo do nó para nulo, sendo que

ele é obrigatório no esquema.

Identificador do Caso de Teste: 008
Objetivo do teste: Tentar alterar um comentário no sistema sem ter permissão de UPDATE ao esquema POST
Resultado esperado: Não permitir instanciar nós quando não possui permissão aquele esquema de nós.
Data da Execução: 20/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe dos testes: // se conectar com um usuário que não tem permissão de UPDATE MATCH (n:POST) where n.ID =1 SET n.Titulo = 'Chuvas no Brasil'
Histórico dos testes 2. SUCESSO: Não foi alterar as propriedades do nó sem a permissão concedida..

Identificador do Caso de Teste: 009
Objetivo do teste: Excluir ua reportagem no sistema sem ter permissão de DELETE ao esquema POST
Resultado esperado: Não permitir excluir nós quando não possui permissão de DELETE ao esquema de nós.
Data da Execução: 20/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe dos testes: // se conectar com um usuário que não tem permissão de DELETE MATCH (n:POST) where n.ID =1 delete n
Histórico dos testes 1. SUCESSO: Não foi possível excluir o nó sem ter permissão.

Identificador do Caso de Teste: 010
Objetivo do teste: Conceder permissão de DELETE ao esquema POST e alterar o nó removendo uma propriedade obrigatória.
Resultado esperado: Não permitir alterar nós de modo que não respeite as regras do esquema.
Data da Execução: 20/12/2015
Resultado: (X) Sucesso / ( ) Falha
Detalhe dos testes: // Alterar um atributo obrigatório para opcional MATCH (n:POST) where n.ID =1 SET n.Titulo = ''
Histórico dos testes 1. ERRO: Foi possível remover o valor da propriedade mesmo ela sendo

obrigatória na definição do esquema.

2. CORREÇÃO:Validar os campos obrigatórios na alteração da instancia.
3. SUCESSO: Não foi possível alterar um atributo do nó para nulo, sendo que ele é obrigatório no esquema.