Comparação entre MySQL e Neo4J para o Acesso a Dados Complexos Usando Linguagens Declarativas

Émerson P. Homrich[1], **Sergio L. S. Mergen**[1]

[1]Universidade Federal de Santa Maria (UFSM)
Santa Maria, RS – Brasil

{ehomrich,mergen}@inf.ufsm.br

Abstract. *Graph databases are a type of NoSQL databases focused on highly connected data and dynamic relationships, characteristics that make them plausible for applications such as social networks and preferences systems. With data becoming increasingly sparse and semi-structured, it's questioned whether graph databases already have maturity to be more advantageous than relational databases to access complex data using queries with a declarative syntax. The purpose of this work is to compare two database leaders in graph and relational technnology (Neo4J and MySQL, respectively) with respect to their performance in accessing complex data, using only the resources of their declarative query languages.*

Resumo. *Os bancos de dados de grafo são um tipo de bancos de dados NoSQL voltados a dados altamente conectados e com relacionamentos dinâmicos, características plausíveis para aplicações como redes sociais e sistemas de preferências. Com os dados se tornando cada vez mais esparsos e semi-estruturados, questiona-se se bancos de dados de grafo já possuem maturidade para serem mais vantajosos do que bancos de dados relacionais para acessar dados complexos usando consultas com uma sintaxe declarativa. O objetivo deste trabalho é comparar dois bancos de dados líderes nas tecnologias de grafo e relacional (Neo4J e MySQL, respectivamente) em relação ao seu desempenho no acesso a dados complexos, usando apenas recursos de suas linguagens declarativas.*

1. Introdução

Bancos de dados de grafo são um tipo de bancos de dados NoSQL voltado para trabalho com dados altamente conectados e com relacionamentos dinâmicos em grandes volumes. Tais características os tornam opções plausíveis para aplicações como redes sociais, sistemas de recomendações e outros tipos de dados complexos [Sadalage and Fowler 2012].

Geralmente construídos para uso com sistemas transacionais, são otimizados para garantir integridade transacional e disponibilidade de operação, assim como os bancos de dados relacionais [Robinson et al. 2013]. Outra característica em comum é o uso de linguagens de consulta declarativas (ex. SQL e Cypher). Muito embora existam interfaces de acesso mais procedurais, as linguagens declarativas aumentam a legibilidade do código, levando a maior manutenibilidade e produtividade.

O crescimento de dados guiados pelo usuário aumentou o volume e o tipo de dados gerados. Em paralelo a esse crescimento, os dados também estão se tornando cada vez

mais esparsos e semi-estruturados [Tiwari 2011]. Com isso, questiona-se se bancos de dados de grafo já possuem maturidade suficiente para serem mais vantajosos em relação à tecnologia dos bancos de dados relacionais para trabalhar com dados complexos usando apenas comandos da linguagem de consulta declarativa.

Nesse sentido, este trabalho tem o intuito de comparar bancos de dados relacionais e bancos de dados não relacionais orientados a grafo, com foco no MySQL e no Neo4j, analisando características e o desempenho desses SGBDs no acesso a dados complexos. A escolha pelos representantes de cada tipo de SGBD foi tomada considerando o ranking de bancos de dados do site *DB-Engines*[1]. O Neo4j ocupa a 21ª posição no ranking geral, envolvendo todos os SGBDs, e a 1ª posição no ranking de bancos de dados de grafo. O MySQL, por sua vez, é o segundo mais popular tanto entre sua categoria como nos bancos de dados em geral.

Este trabalho está organizado da seguinte forma: a seção 2 apresenta trabalhos correlatos. A seção 3 apresenta uma comparação entre características dos modelos relacional e de grafos. Na seção 4 são apresentados os testes realizados e os resultados obtidos. Por fim, a seção 5 trata das conclusões e considerações finais.

2. Trabalhos Relacionados

Considera-se como trabalhos relacionados aqueles que analisam o desempenho de operações nos SGBDs MySQL e Neo4j com abordagens semelhantes à proposta deste trabalho. Neste contexto, são discutidos alguns dos trabalhos já realizados.

Em [Batra and Tyagi 2012], é realizado um experimento que consiste em um conjunto de consultas explorando relacionamentos. O domínio de dados é composto por usuários com relações de amizade, e cada usuário pode ter filmes favoritos estrelados por atores. Um conjunto de consultas complexas foi definido, como a busca pelos protagonista dos filmes favoritos dos amigos de um usuário. As consultas foram realizadas sobre bases com 100 e 500 usuários, para verificar o comportamento dos SGBDs com o crescimento do volume de dados. Os resultados mostram a prevalência do Neo4j nos testes, com tempos de execução consideravelmente menores que os do MySQL e pouca variação com o aumento da base de dados.

De modo semelhante, [Medhi and Baruah 2017] apresenta um experimento de desempenho. O domínio aborda jogadores, times e partidas de críquete. As consultas foram realizadas recuperando 100, 300 e 400 objetos. O desempenho do Neo4j prevalece nos testes, sendo duas vezes mais rápido nos bancos de dados com 100 objetos e até dez vezes mais rápido quando o número de objetos sobe para 400.

Em [Vicknair et al. 2010], são utilizados grafos acíclicos direcionados gerados aleatoriamente. No experimento, utilizou 4 grafos, de 1.000, 5.000, 10.000 e 100.000 nós. Os nós de cada grupo guardam um atributo inteiro aleatório ou cadeia de caracteres de 8KB ou 32KB. Constatou-se que o tamanho dos grafos no Neo4j, em MB, pode ser até duas vezes maior que os grafos no MySQL. Os tempos de execução das consultas estruturais foram menores no MySQL com os grafos menores e/ou de nós com atributos inteiros, mas consideravelmente mais rápidos no Neo4j quando realizadas sobre os grafos maiores ou de nós com atributos de caracteres. Nas consultas de dados, o MySQL apre-

[1]https://db-engines.com

sentou desempenho muito superior ao Neo4j utilizando campos numéricos. O contrário ocorreu em consultas com parâmetros de texto. O ambiente de testes foi uma máquina com 4GB RAM e CPU Intel(R) Core(TM) 2 Duo com 3GHz de frequência de operação.

Com exceção de [Vicknair et al. 2010], os demais trabalhos utilizaram bases de dados bastante reduzidas durante os experimentos. Os trabalhos também não realizaram as consultas de forma padronizada, isto é, não utilizaram um mesmo meio para realizá-las. [Batra and Tyagi 2012] e [Medhi and Baruah 2017] realizam as consultas do MySQL através de *scripts* em linguagem PHP e as consultas do Neo4j diretamente em linguagem Cypher. A falta de padrão no método pode ter exposto as consultas ao MySQL à influência da linguagem.

3. O Modelo Relacional e o Modelo de Grafos

A eficiência no acesso aos dados pode sofrer forte influência da forma como os dados são modelados, bem como das interfaces de acesso disponibilizadas. Essa seção resume as principais diferenças entre o modelo relacional e o modelo orientado a grafos nesses dois aspectos.

No modelo relacional, o banco de dados é um conjunto de relações [Ramakrishnan and Gehrke 2008]. Já nos bancos de dados de grafo, o modelo baseia-se na teoria dos grafos. Os dados são representados como um conjunto de vértices (nós) conectados por arestas (relacionamentos), sendo que ambos podem conter propriedades (atributos) no formato de chave-valor. Os nós representam objetos e as arestas representam a forma como esses objetos se relacionam, formando caminhos [Robinson et al. 2013]. Nós podem ser agrupados em rótulos e relacionamentos podem ter nomes e devem possuir uma direção.

Diferente do modelo relacional que é regrado pelo esquema (e que por vezes é considerado rígido por essa característica), o modelo de grafos possui formato livre. Nós de um mesmo rótulo podem ter estruturas diferentes. Outra diferença é o formato dos relacionamentos. No modelo relacional, os relacionamentos são feitos através de campos de chave estrangeira, e podem exigir a criação de tabelas intermediárias. No modelo de grafos, os relacionamentos são tão importantes quanto os nós em si e são armazenados como relacionamentos de fato [Sadalage and Fowler 2012]. Os nomes dos relacionamentos adicionam clareza semântica à estruturação dos nós [Robinson et al. 2013].

No que tange à interface de acesso, ambos os SGBDs adotam linguagens de consultas declarativas. Os bancos de dados relacionais utilizam a linguagem *Structured Query Language* (SQL) [Silberschatz et al. 2010]. O Neo4j utiliza a linguagem Cypher[2], inicialmente desenvolvida para uso exclusivo e posteriormente adotada por outros bancos de dados de grafo através do projeto openCypher. É uma linguagem compacta e expressiva, projetada para ser de fácil leitura, e seu uso é de acordo com a forma intuitiva na qual os grafos geralmente são feitos em diagramas [Robinson et al. 2013].

Tanto nos bancos de dados relacionais como no Neo4j, a execução de consultas constitui-se por etapas. As consultas SQL passam por verificação e validação, que analisa a sintaxe da consulta e identifica as tabelas envolvidas, bem como os atributos solicitados

[2]Existem outras linguagens de consulta amplamente suportadas pelos bancos de dados de grafo (inclusive pelo Neo4j), como a *SPARQL Protocol and RDF Query Language* (SPARQL) e a Gremlin.

ou utilizados para filtragens. O SGBD traduz a consulta para uma estrutura como uma árvore de consulta, e define uma estratégia de execução, que é decomposta em blocos. A escolha da estratégia de execução é a complexa tarefa de otimização de consulta, que nem sempre será a melhor, e sim a mais razoável [Elmasri and Navathe 2011].

Na linguagem Cypher, as consultas também são executadas através de planos de execução. Cada consulta é dividida em porções menores chamadas de operadores[3]. O conteúdo da consulta é tokenizado e avaliado semanticamente, formando uma árvore sintática abstrata após ser otimizado e normalizado. Um grafo de consulta é criado, e então são definidos planos lógicos. A seletividade dos rótulos e índices, bem como a cardinalidade de registros são definidos baseados em estatísticas. A partir dos planos lógicos, um algoritmo seleciona o plano menos custoso e otimiza-o para gerar o plano de execução [4].

Consultas a dados conectados são realizadas de forma diferente nos dois SGBDs. No MySQL (e nos demais bancos de dados relacionais), os relacionamentos são buscados através de junções, combinando registros em tempo de recuperação através de regras, como igualdade de campos em registros. No Neo4j, é feito o processo de travessia, que consiste em seguir os caminhos existentes no grafo levando em consideração a direção dos relacionamentos e suas propriedades [Robinson et al. 2013].

No Neo4j, também é possível realizar travessias em nível de comprimento. Esse tipo de travessia consiste em seguir um padrão de caminho no grafo, contendo muitos nós e relacionamentos em sequência. A linguagem Cypher disponibiliza recursos sintáticos chamados de comprimento variável para realizar a travessia através dos relacionamentos[5]. Considera-se cada relacionamento acessado como um novo nível, e esse processo é realizado de forma recursiva.

A Figura 1 apresenta exemplos de consultas em Cypher. A filtragem é realizada através da cláusula *WHERE*, comum à SQL. Nas consultas de travessia, o recurso de comprimento variável pode ser utilizado adicionando as cardinalidades após o relacionamento a ser explorado, no formato <*min*>..<*max*>. Na Figura, a segunda consulta explora o relacionamento *CONNECTED_TO* em três níveis, declarando *[:CONNECTED_TO*..3]*. O nível mínimo foi omitido, indicando que a travessia deve considerar desde a origem.

FILTRAGEM	COMPRIMENTO VARIÁVEL

`MATCH (a:Intersection)`
`WHERE a.num_hotels = 5`
`RETURN a.intersection_id`
	`MATCH (a:Intersection)-`
	`[:CONNECTED_TO*1..3]->(b:Intersection)`
	`WHERE a.intersection_id = 213`
	`RETURN DISTINCT b.intersection_id`

Figura 1. Exemplos de consultas com filtragem e travessia em Cypher

A linguagem Cypher pode ser limitada para alguns cenários de consultas que exijam muito processamento durante a expansão do subgrafo, sendo necessária a utilização de bibliotecas de *procedures* como a APOC ou a implementação de *procedures* através da Traversal API em Java do Neo4j. A segunda opção oferece maior liberdade de escolha

[3]https://neo4j.com/docs/developer-manual/current/cypher/execution-plans/
[4]https://neo4j.com/blog/tuning-cypher-queries/
[5]https://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/#cypher-pattern-varlength

nas decisões do processamento, mas a implementação de *procedures* é consideravelmente mais complexa do que as consultas escritas de forma puramente declarativa.

4. Testes e Resultados

Esta seção apresenta experimentos realizados para comparar o desempenho dos SGBDs MySQL e Neo4j em tarefas como a carga de dados e alguns cenários de consulta, com foco na utilização de apenas recursos padrão das linguagens de acesso dos SGBDs. O objetivo principal dos experimentos é verificar se há vantagem em utilizar bancos de dados de grafo em detrimento de bancos de dados relacionais para o acesso a dados complexos.

4.1. Ambiente de testes

Os testes foram realizados através um *framework* desenvolvido em linguagem Python, versão 3.6.4. O uso do *framework* visa padronizar o acesso aos dois SGBDs e tornar os testes simétricos.

Os experimentos foram realizados sobre uma máquina dispondo de processador Intel(R) Core(TM) i5-3230M, de terceira geração. O processador dispõe de 4 núcleos (2 núcleos reais e 2 simulados, resultando em 4 *threads*), frequência de operação de 2,6GHz, 3MB de memória cache e placa de vídeo compartilhada HD Graphics 4000. A máquina dispõe de 6GB de memória RAM DDR3 com frequência de 1600 MHz.

O sistema operacional utilizado na máquina foi o Linux Mint 18.3, edição Cinnamon 64 bit. O sistema operacional e os processos nativos ocupam aproximadamente 1,2GB de memória RAM, deixando em torno de 4,5GB de memória disponíveis para a realização dos testes. Não foram utilizadas máquinas virtuais, pois a quantidade de memória disponível seria consideravelmente reduzida. Além disso, a máquina virtual sofreria influência do sistema hospedeiro, e a execução dos testes também sofreria influência do sistema operacional instalado na mesma.

4.2. Bancos de dados e ferramentas utilizadas

As funcionalidades disponíveis de cada SGBD foram analisadas para decidir quais seriam utilizadas e se alguma configuração de ambiente precisava ser alterada. No MySQL, optou-se por utilizar duas das *engines* disponíveis: InnoDB e MyISAM. A *engine* InnoDB foi escolhida pelo fato de implementar todas as propriedades ACID e por ser o padrão do MySQL. Já MyISAM foi escolhida por ser simplificada e geralmente recomendada para dados mais utilizados para leitura do que escrita.

Duas versões do MySQL foram utilizadas: 5.7.21 e 8.0.4-rc. O Neo4j foi utilizado em sua versão a 3.3.2. A conexão aos SGBDs foi realizada por conectores para linguagem Python mencionados em seus guias e recursos para desenvolvedores[6][7]. Para o MySQL, utilizou-se o conector oficial, MySQL Connector/Python, e para o Neo4j, o conector Py2neo.

4.3. Definição do domínio de dados

Para a realização do estudo, foi utilizado o grafo da rede rodoviária do estado da Califórnia, proveniente do *Stanford Network Analysis Project* (SNAP)[8]. O grafo foi escolhido

[6]https://www.mysql.com/products/connector/
[7]https://neo4j.com/developer/python/
[8]https://snap.stanford.edu

por ser um domínio de dados real, com quantidade significativa de nós e arestas e com amplo uso de relacionamentos.

Neste grafo, interseções e pontos de extremidade são representados por nós, enquanto as ruas conectando essas interseções ou pontos de extremidade da estrada são representadas por arestas (relacionamentos) não direcionadas [Leskovec and Krevl 2014]. A base de dados possui 1.965.206 nós e 2.766.607 relacionamentos. Os nós e relacionamentos não possuem nenhum atributo.

4.4. Pré-processamento

A base de dados foi processada para que os nós fossem extraídos. Dados fictícios com valores pseudoaleatórios foram adicionados aos nós para permitir consultas com filtragem. Os nós receberam três novos atributos: *num_hotels*, que representa número de hotéis, *num_restaurants*, como o número de restaurantes e *num_gas_stations*, que representa número de postos. Os valores desses atributos foram gerados através de sorteio ponderado.

A Tabela 1 apresenta os possíveis valores dos novos atributos, com seus respectivos pesos. Os pesos totalizam 100% e foram divididos de modo que alguns valores permitissem alta seletividade quando utilizados ou combinados em filtros.

Tabela 1. Valores possíveis e pesos para sorteio nos atributos fictícios adicionados aos nós

Valor	0	1	2	3	4	5
Peso	30%	30%	20%	14%	5%	1%

4.5. Modelagem

No MySQL, o grafo foi representado como um relacionamento muitos para muitos, fazendo uso de duas tabelas, considerando que múltiplas arestas podem partir de um nó ou chegar até o mesmo simultaneamente. Os nós foram representados por uma tabela *Intersection*, contendo os três atributos adicionados no pré-processamento e o identificador do nó, *intersection_id*, como chave primária. As ruas foram representadas por uma tabela *Road*, que contém referências aos identificadores dos nós de partida e chegada. Essa estrutura pode ser observada na Figura 2.a).

No Neo4j, apesar da ausência de esquema, os nós seguem a mesma estrutura e nome da tabela *Intersection* do MySQL, enquanto os relacionamentos foram nomeados *CONNECTED_TO*. O Neo4j adiciona, por padrão, um atributo *id* imutável em todos os nós criados. Para evitar quaisquer problemas de identificação, foi criado o *intersection_id* nos nós. Uma prévia da estrutura do Neo4j é representada na Figura 2.b).

4.6. Metodologia dos testes

Para garantir a homogeneidade dos testes, algumas medidas foram tomadas para reduzir o impacto do sistema operacional e para restringir o uso de cache apenas no contexto de cada banco de dados. Uma das medidas foi o encerramento dos processos não nativos da máquina de modo a garantir a maior quantidade de memória livre.

Os testes de carga de dados foram realizados uma vez cada, com apenas um SGBD ativo por vez. Para os testes onde uma única operação exige múltiplas transações, o

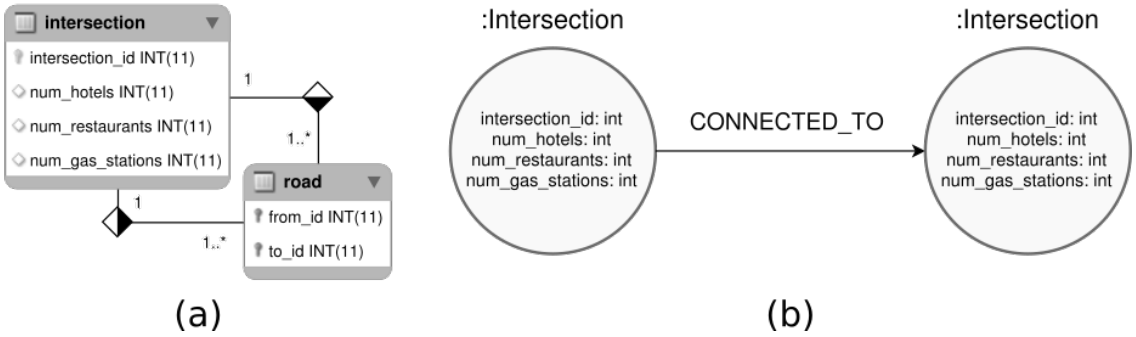

Figura 2. Representação da estrutura do grafo no MySQL e no Neo4j

tempo total de execução é o somatório do tempo decorrido em cada transação. Nas de dados, cada teste foi executado trinta vezes no MySQL e no Neo4j, de forma intercalada. Em cada iteração, todas as consultas são executadas nas *engines* MyISAM e InnoDB e no Neo4j, nessa ordem. O resultado de cada consulta é a média aritmética dos tempos coletado nas trinta execuções, excluindo os 10% melhores e os 10% piores resultados.

A versão 5.7.21 do MySQL foi utilizada em todos os testes, exceto os testes de consultas por caminhos onde nós satisfazem uma condição. Neste cenário de teste, foi utilizado o MySQL 8.0.4-rc, que possui a implementação de funções recursivas (*WITH RECURSIVE*). Estes testes foram executados separados dos demais, mas seguindo a organização de número de execuções, cômputo de resultados e execuções intercaladas.

4.7. Carga de dados

A carga é importante em cenários onde se deseja armazenar os dados complexos temporariamente em um SGBD para resolver problemas pontuais que demandem maior processamento. A carga foi realizada de duas formas: a inserção tradicional e a importação de dados de arquivos.

A inserção tradicional foi feita através dos comandos *INSERT INTO* e *CREATE*, comandos padrão de inserção de registros das linguagens SQL e Cypher, respectivamente. A importação por arquivos foi realizada através de recursos das linguagens SQL e Cypher que permitem que o conteúdo de arquivos sejam processados e inseridos nos bancos de dados. Em ambos os casos, foram tomadas medidas de otimização para acelerar as cargas.

No MySQL, foram desativadas as verificações de singularidade e de chave estrangeira, além da desativação do *commit* automático. No Neo4j, foram criados índices no atributo identificador de cada nó anterior à inserção dos dados, pois o SGBD usa propagação em segundo plano para concluir a operação. A criação dos índices é de suma importância uma vez que no Neo4j, é necessário buscar os nós envolvidos em um relacionamento para poder criá-lo.

A inserção tradicional nos dois SGBDs foi realizada em partes, com transações de 40.000 objetos cada. A quantidade de objetos por transação foi escolhida de forma empírica. A medida se fez necessária por limitações de memória para o processamento do Neo4j, e foi replicada nas inserções no MySQL para garantir homogeneidade nos testes. Para a importação, a mesma quantidade de objetos por transação foi utilizada no Neo4j. Já a SQL não permite esse tipo de quebra transacional.

A Tabela 2 apresenta os resultados dos testes, em segundos. A *engine* MyISAM no MySQL apresentou o melhor desempenho em todos os casos. O desempenho do Neo4j foi inferior aos do MySQL em qualquer sentido, com inserções consideravelmente mais dispendiosas mesmo na importação, com comandos otimizados para inserção massiva.

Tabela 2. Desempenho das cargas de dados no MySQL e no Neo4j

	Importação		Inserção	
	Nós	Relacionamentos	Nós	Relacionamentos
MySQL MyISAM	3,73 s	7,05 s	442,41 s	576,25 s
MySQL InnoDB	19,04 s	60,95 s	451,94 s	629,54 s
Neo4j	73,79 s	103,09 s	756,61 s	860,89 s

4.8. Consultas com filtragem de dados

A segunda modalidade de testes foi a de consultas com filtragens de dados. Os testes desta modalidade foram divididos em aplicações de filtros de igualdade sobre um e dois atributos indexados. Para as consultas com um filtro, buscou-se por nós com valor 5 no campo *num_hotels*, que correspondem a 1% da base de dados. Nas consultas com dois atributos, foram utilizados *num_restaurants*, com valor igual a 2, e *num_gas_stations*, com valor 4, combinados com o operador lógico *AND*. Os valores escolhidos representam 20% e 5% dos nós ou registros, respectivamente, e combinados também recuperam em torno de 1% da base de dados.

Os resultados são apresentados pela Tabela 3. O Neo4j apresentou desempenhos inferiores aos obtidos pelas *engines* do MySQL, com tempo médio de execução próximo de um segundo. A *engine* InnoDB apresentou o melhor desempenho para o MySQL, com execução aproximadamente três vezes mais rápida que a de MyISAM na aplicação de um filtro. Os resultados utilizam seis casas decimais em sua representação, devido às diferenças insignificantes entre os resultados das *engines* do MySQL.

Tabela 3. Desempenho das consultas com filtragem de dados com índice

	Um filtro	Dois filtros
MySQL MyISAM	0,002980 s	0,011908 s
MySQL InnoDB	0,001191 s	0,016102 s
Neo4j	0,885031 s	0,936510 s

4.9. Consultas com cruzamento de dados

A terceira modalidade de testes explora os relacionamentos no grafo. Para isso, foram executadas consultas que fazem uso da cláusula *JOIN* no SGBD MySQL e a travessia de caminhos no Neo4j. Foram testados dois cenários de consulta explorando a vizinhança de um nó específico, buscando por todos os nós em até três níveis de relacionamentos a partir do mesmo, e todos os nós a exatos três níveis de relacionamento.

No Neo4j, foi utilizado o recurso de comprimento variável. No MySQL, o comportamento foi simulado utilizando diversas junções na tabela de relacionamentos (*Road*). Os resultados podem ser observados na Tabela 4. A *engine* InnoDB e o Neo4j apresentaram os melhores desempenhos, em ordem, com tempos de execução foram na faixa dos

milissegundos. A *engine* InnoDB obteve um desempenho mais satisfatório que o Neo4j, sendo aproximadamente sete vezes mais rápida, ainda que esperava-se vantagem do Neo4j nesses cenários. O tempo de execução da *engine* MyISAM manteve-se constante.

Tabela 4. Desempenhos das consultas com cruzamento de dados

	Consulta de nós em até três níveis	Consulta de nós a três níveis
MySQL MyISAM	11,540125 s	11,522095 s
MySQL InnoDB	0,001099 s	0,000870 s
Neo4j	0,007750 s	0,006132 s

4.10. Consultas por caminhos cujos nós satisfazem uma condição

O último teste proposto consiste em uma busca de comprimento variável com condições baseadas em propriedades dos nós. Diferentemente dos testes de travessia anteriores, onde retornava-se todos os nós envolvidos no níveis compreendidos pela consulta, este cenário busca apenas pelos caminhos onde todos os nós obedecem uma determinada condição sobre um dos seus atributos, dado um nó de partida.

As consultas construídas em SQL utilizam funções recursivas (*WITH RECURSIVE*), disponíveis na versão 8.0.4-rc do MySQL, pois a recursão é necessária para que o comprimento do caminho possa ser parametrizável. As consultas realizam junções entre as tabelas *Intersection* e *Road*, e utilizam uma coluna pivô *n*, incrementada a cada chamada recursiva, como critério de parada. As consultas em Cypher, por sua vez, continuam a usar comprimento variável, agora aplicando um filtro sobre os nós de cada caminho.

Foi aplicado um filtro de igualdade sobre o atributo *num_hotels*, verificando quais caminhos existem com nós cujo valor do atributo é menor ou igual a 2. Além disso, a consulta foi executada múltiplas vezes fazendo uso de três comprimentos de caminho: 4, 6 e 8 relacionamentos.

A Tabela 5 apresenta os resultados obtidos. O MySQL apresentou tempos de execução superiores ao Neo4j, com a *engine* InnoDB obtendo o melhor desempenho no geral. Os tempos de execução do Neo4j foram de três a cinco vezes maiores que os tempos da *engine* InnoDB, dependendo do comprimento do caminho analisado. Os tempos de execução dos dois SGBDs apresentaram variação conforme o aumento do comprimento.

Tabela 5. Desempenho das consultas por caminhos com condição

	Comprimento do caminho		
	4	6	8
MySQL MyISAM	0,001931 s	0,002123 s	0,002559 s
MySQL InnoDB	0,001743 s	0,001999 s	0,002400 s
Neo4j	0,006079 s	0,008911 s	0,012885 s

5. Conclusões

Este trabalho teve a finalidade de fornecer uma visão geral a respeito do modelo relacional e do modelo de grafos, com foco no MySQL e no Neo4j, e uma comparação de desempenho em operações comuns, como carga e alguns tipos de consulta de dados.

A constatação que se faz a partir dos resultados obtidos é que os bancos de dados de grafo ainda não atingiram maturidade de recursos suficiente para superarem a eficiência dos bancos de dados relacionais ao trabalhar com dados de alta complexidade utilizando apenas suas linguagens declarativas.

O Neo4j oferece uma linguagem de acesso simples e intuitiva para grafos, com recursos predefinidos para manipulação dos dados durante a expansão dos subgrafos nas consultas. É possível que essa simplicidade seja um dos motivos para sua popularidade em detrimento das demais opções de sua categoria. Entretanto, os bancos de dados relacionais também permitem o acesso a dados complexos de forma relativamente simples. As funções recursivas das expressões de tabela comum em SQL são um exemplo de recursos tão ou até mais sofisticados que o comprimento variável da linguagem Cypher.

Mesmo em casos em que seja aceitável o uso de recursos procedurais de linguagens orientadas a grafos, pode ser mais vantajoso recorrer a outra abordagem, tanto em termos de eficiência quanto de familiaridade com a linguagem. Uma possibilidade é o uso de critérios de seleção mais abrangentes no SGBD e a transferência da complexidade do tratamento para a aplicação. Este trabalho serve como ponto de partida para que questões como essa venham a ser discutidas.

Referências

Batra, S. and Tyagi, C. (2012). Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(2):509–512.

Elmasri, R. and Navathe, S. B. (2011). *Sistemas de Banco de Dados*. Pearson Addison Wesley, São Paulo, 6 edition.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection.

Medhi, S. and Baruah, H. K. (2017). Relational database and graph database: A comparative analysis. *Journal of Process Management. New Technologies*, 5(2):1–9.

Ramakrishnan, R. and Gehrke, J. (2008). *Sistemas de Gerenciamento de Banco de Dados*. McGraw-Hill, São Paulo, 3 edition.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media, Inc.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1 edition.

Silberschatz, A., Korth, H. F., and Sudarshan, S. (2010). *Database System Concepts*. McGraw-Hill, 6 edition.

Tiwari, S. (2011). *Professional NoSQL*. Wrox Press Ltd., Birmingham, UK, UK.

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6. ACM.