AVALIAÇÃO COMPARATIVA DE DESEMPENHO EM BIG DATA: Modelos Relacionais (MySQL) Versus Bancos de Dados de Grafos (Neo4j)

CLARISSA ERI MORITA[1] **, GIOVANNA DORNELLES BARICHELLO**[1]
ISABELLA VICENTE[1] **, JOÃO PEDRO QUEIROZ DEGER**[1]
PETERSON ALMEIDA FONTINHAS[1]

[1]Universidade Federal do Paraná (UFPR)
Curitiba, PR, Brasil

clarissa.morita@ufpr.br , giovannadornelles@ufpr.br , isabellavicente@ufpr.br

2

joao.deger@ufpr.br , peterson.fontinhas@ufpr.br

Abstract. *This paper presents a comparative performance evaluation between a traditional Relational Database Management System (RDBMS), represented by MySQL, and a specialized NoSQL Graph Database, Neo4j, when dealing with a highly interconnected Big Data social graph dataset. Using the MUSAE GitHub Social Network dataset, the study investigates operational efficiency across aggregation, textual, and, crucially, relationship queries (graph traversal). Results demonstrate that for deep traversal operations (degree* > 2*), Neo4j achieves orders of magnitude superior performance compared to the quadratic complexity cost of MySQL, proving the decisive advantage of the native graph architecture for network analysis in Big Data contexts.*

Resumo. *Este artigo apresenta uma avaliação comparativa de desempenho entre um Sistema Gerenciador de Banco de Dados Relacional (SGBDR), representado pelo MySQL, e um Banco de Dados de Grafos NoSQL, Neo4j, no contexto da manipulação de um dataset de Big Data com alta interconexão (grafo social). O estudo utiliza o dataset MUSAE GitHub Social Network para investigar a eficiência operacional em consultas de agregação, textuais e, crucialmente, de relações (travessia de grafo). Os resultados demonstram que, em operações de profundidade (grau* > 2*), o Neo4j oferece ganhos de desempenho que chegam a ordens de magnitude em comparação com o custo de complexidade quadrática do MySQL, provando a superioridade da arquitetura de grafo nativa para análises de redes em Big Data.*

1. Introdução

A explosão no volume, velocidade e variedade de dados (Big Data) evidenciou as limitações dos SGBDRs na manipulação de estruturas de **alta cardinalidade e relações complexas** [**?**]. Para enfrentar isso, surgiram os SGBDs NoSQL, sendo os **Bancos de Dados de Grafos** cruciais para a análise de redes sociais.

1.1. Apresentação do Problema

O problema central deste trabalho reside na **avaliação comparativa do desempenho** entre um SGBD Relacional tradicional e um SGBD NoSQL especializado ao manipular um dataset com características de Big Data de grafo.

Especificamente, investigamos:

- A capacidade de modelagem e representação da complexidade de um grafo social em um modelo tabular (MySQL) versus um modelo de grafo (Neo4j).
- A eficiência operacional de ambos os sistemas em três categorias cruciais de consultas (Agregação, Texto e Relação), buscando quantificar o custo de processamento em cada modelo.

Neste estudo, o MySQL foi escolhido para representar a tecnologia Relacional, e o Neo4j, um SGBD de Grafos nativo, foi selecionado como o representante da tecnologia Big Data, utilizando o dataset MUSAE GitHub Social Network como estudo de caso.

1.2. Objetivos do Trabalho

O objetivo principal deste projeto é **avaliar e demonstrar as diferenças, aplicações e limitações** dos sistemas Relacionais (MySQL) em comparação com os sistemas não-Relacionais (Neo4j) no contexto do gerenciamento e manipulação de Big Data.

Os objetivos específicos a serem alcançados incluem:

1. **Modelagem e Carga de Dados:** Implementar a estrutura de dados do MUSAE GitHub Social Network tanto no MySQL quanto no Neo4j, e registrar o desempenho da carga inicial em ambas as ferramentas.
2. **Execução de Consultas:** Desenvolver e executar um conjunto de consultas estratégicas (Buscas de Agregação, de Texto e de Relação) em ambas as plataformas.
3. **Coleta e Análise de Desempenho:** Medir e tabular as métricas de desempenho (tempo de execução, uso de recursos) para cada tipo de consulta, focando em operações que exigem travessia de grafo (busca de vizinhos de 2º grau e caminho mais curto).
4. **Conclusão e Recomendação:** Comprovar a hipótese de que o Neo4j oferece ordens de magnitude de desempenho superior em operações de grafo complexas, fornecendo uma base técnica para a escolha da tecnologia em projetos de Big Data orientados a relacionamentos.

2. Tecnologia e Modelo de Grafo

2.1. O Modelo de Grafo de Propriedades

A tecnologia NoSQL escolhida, o **Neo4j**, utiliza o **Modelo de Grafo de Propriedades** (*Property Graph Model*). Este modelo se distingue do relacional por tratar as **relações (conexões)** como dados de primeira classe. Os dados são representados por **Nós** (entidades, rotulados e com propriedades) e **Relações** (conexões direcionais com propriedades).

Esta arquitetura garante que o custo para atravessar (navegar) de um nó para seus vizinhos seja **constante** ($O(1)$) e não dependa do tamanho total do banco de dados. Isso

contrasta com o SGBDR, onde a busca de vizinhos de k-ésimo grau exige k operações de `JOIN` sucessivas, com um custo próximo a $O(n^k)$, que cresce exponencialmente [**?**].

As principais vantagens do modelo de grafo em aplicações como Redes Sociais, Detecção de Fraudes e Gerenciamento de Identidade podem ser visualizadas na Figura 1.

3. Estudo de Caso e Adequação do Dataset

3.1. Dataset MUSAE GitHub Social Network

Utilizamos o dataset **MUSAE GitHub Social Network** [**?**]. Este conjunto de dados é a escolha ideal para esta avaliação comparativa, pois ele é **intrinsecamente um grafo**, modelando uma rede social colaborativa. A estrutura do dataset (Figura 2) é definida por: $\approx$ 37.700 **Nós** (Usuários do GitHub) e $\approx$ 290.000 **Arestas** (Conexões de seguidores/colaboração).

A **alta cardinalidade e complexidade das relações** garantem que qualquer consulta que envolva a descoberta de caminhos ou vizinhos de 2º grau forçará o SGBDR (MySQL) a executar múltiplos e custosos `JOIN`s, expondo a limitação do modelo tabular.

3.2. Modelagem

A modelagem no MySQL seguiu a 3NF. No Neo4j, usamos nós (`:User`) e relações (`:FOLLOWS`).

4. Resultados e Avaliação de Desempenho

Os experimentos compararam o tempo de execução (em milissegundos) para as consultas em ambas as plataformas.

4.1. Buscas de Agregação e Texto

Em agregação, o desempenho do Neo4j em encontrar os 10 maiores conexões (Figura 3) e os 5 menores (Figura 4) mostra que o Neo4j é significativamente mais rápido.

Em buscas textuais, a eficiência de `CONTAINS` (Figura 5) e `REGEXP` (Figura 6) é largamente superior no Neo4j.

4.2. Buscas de Relação (Traversal)

O desempenho diverge dramaticamente nas operações de travessia, conforme detalhado nas figuras de resultados:

1. **Vizinhos de 1º Grau:** O Neo4j foi instantâneo, conforme ilustra a Figura 7.
2. **Vizinhos de 2º Grau:** O Neo4j manteve o tempo baixo (Figura 8), uma superioridade de **2 a 3 ordens de magnitude** sobre o MySQL.
3. **Caminho Mais Curto (*Shortest Path*):** O Neo4j, com o algoritmo nativo (Figura 9), executou a tarefa em **5ms a 200ms**, contra **acima de 10 segundos** do MySQL.

5. Conclusão

O estudo comprovou a **superioridade técnica e operacional do Neo4j** em cenários de Big Data orientados a relacionamentos. Enquanto o MySQL falha em operações de profundidade ($k > 1$) devido ao custo $O(n^k)$, o Neo4j mantém um custo linear e previsível. O modelo de grafo é a tecnologia mais adequada e escalável para análise de redes, sendo recomendado para projetos de Big Data que dependem da interconexão dos dados.

Agradecimentos

Agradecemos ao professor: João Eugenio Marynowski pela orientação e infraestrutura de pesquisa.

Referências

6. Anexos de Resultados e Consultas

Aplicação	Problema Resolvido	Vantagem sobre o Relacional
Redes Sociais e Recomendação	Encontrar "amigos de amigos" ou recomendar produtos com base em conexões.	Navegação profunda é instantânea. O *traversal* nativo substitui múltiplos e custosos JOINs.
Detecção de Fraudes	Identificar anéis de fraude (círculos de contas falsas ou conexões suspeitas).	A busca de padrões (e.g., *loops* de 3 a 5 nós) é extremamente eficiente, revelando anomalias invisíveis ao SQL.
Gerenciamento de Identidade	Mapear permissões de acesso e hierarquias complexas.	O Neo4j processa a complexidade de hierarquias M:N (muitos-para-muitos) sem a limitação de profundidade do SQL.

Figura 1. Vantagens do Banco de Dados de Grafos (Neo4j) em Aplicações de Rede.

Tabela	Conteúdo	Estrutura Relacional	Volume Estimado
musae git target.csv (nodes_target)	Nós (usuários do GitHub) estamos metadados.	Tabela principal de usuários.	≈ 37.700$ registros
musae_git_edges. csv (edges)	Arestas (conexões/seguidor es/colaborações) entre os usuários.	Tabela de Relação (arestas), representando o grafo.	≈ 290.000$ registros
musae git features.csv (features)	Atributos/Features associados a cada usuário.	Tabela N:M (características), para buscas baseadas em atributos.	≈ 2.5$ Milhões de registros

Figura 2. Estrutura e Volume do Dataset MUSAE GitHub Social Network.

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT id_1, COUNT(*) AS num_connections FROM edges GROUP BY id_1 ORDER BY num_connections DESC LIMIT 10;	Entre 50ms e 250ms
Neo4j	MATCH (u:User)-[r:FOLLOWS]-> () RETURN u.name, count(r) AS followsCount ORDER BY followsCount DESC LIMIT 10;	Entre 10ms e 30ms

Figura 3. Comparativo de Desempenho em Busca de Agregação (TOP 10).

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT id_1, COUNT(*) AS num_connections FROM edges GROUP BY id_1 ORDER BY num_connections ASC LIMIT 5;	Entre 50ms e 250ms
Neo4j	MATCH (u:User)-[r:FOLLOWS]-> () RETURN u.name, count(r) AS followsCount ORDER BY followsCount ASC LIMIT 5;	Entre 10m e 30ms

Figura 4. Comparativo de Desempenho em Busca de Agregação (BOTTOM 5).

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT * FROM nodes_target WHERE name LIKE '%git%';	Entre 150ms e 800ms
Neo4j	MATCH (u:User) WHERE u.name CONTAINS 'git' RETURN u.name;	Entre 2ms e 15ms

Figura 5. Comparativo de Desempenho em Busca de Texto (LIKE vs CONTAINS).

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT * FROM nodes_target WHERE name REGEXP '^A.*Z$';	Entre 300ms e 1,2s
Neo4j	MATCH (u:User) WHERE u.name =~ 'A.*Z' RETURN u.name;	Entre 10ms e 50ms

Figura 6. Comparativo de Desempenho em Busca de Padrão Complexo (REGEXP).

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT T2.name FROM edges AS T1 JOIN nodes_target AS T2 ON T1.id_2 = T2.id WHERE T1.id_1 = 12345;	Entre 5ms e 50ms
Neo4j	MATCH (:User {id: 12345})-[:FOLLOWS]->(friend:User) RETURN friend.name;	Menos que 5ms

Figura 7. Comparativo de Desempenho em Busca de Relação (1º Grau).

Ferramenta	Consulta	Tempo de execução
MySQL	SELECT DISTINCT T3.name FROM edges AS T1 JOIN edges AS T2 ON T1.id_2 = T2.id_1 JOIN nodes_target AS T3 ON T2.id_2 = T3.id WHERE T1.id_1 = 12345 AND T3.id != T1.id_1 AND T3.id NOT IN (SELECT id_2 FROM edges WHERE id_1 = 12345);	Entre 500ms e 5s
Neo4j	MATCH (u:User {id: 12345})-[:FOLLOWS]->()-[:FOLLOWS]->(friendOfFriend:User) WHERE u <> friendOfFriend RETURN DISTINCT friendOfFriend.name;	Entre 10ms e 50ms

Figura 8. Comparativo de Desempenho em Busca de Relação (2º Grau).

7. Anexos de Resultados e Consultas

Ferramenta	Consulta	Tempo de execução
MySQL	WITH RECURSIVE caminho AS (SELECT id_2 AS usuario, 1 AS distancia, CAST(CONCAT(id_1, ',', id_2) AS CHAR(1000)) AS path FROM edges WHERE id_1 = 0 UNION ALL SELECT e.id_2, c.distancia + 1, CONCAT(c.path, ',', e.id_2) FROM caminho c JOIN edges e ON c.usuario = e.id_1 WHERE c.distancia < 15 AND FIND_IN_SET(e.id_2, c.path) = 0 AND c.usuario != 1) SELECT 0 AS usuario_origem, (SELECT name FROM users WHERE id = 0) AS nome_origem, 1 AS usuario_destino, (SELECT name FROM users WHERE id = 1) AS nome_destino, MIN(distancia) AS distancia_minima FROM caminho WHERE usuario = 1;	mais de 10s (para mais de 3 graus)
Neo4j	MATCH (start:User {id: 0}) MATCH (end:User {id: 1}) WITH start, end MATCH path = shortestPath((start)-[:FOLLOWS*]-(end)) RETURN start.id AS usuario_origem, start.name AS nome_origem, end.id AS usuario_destino, end.name AS nome_destino, length(path) AS distancia_minima;	Entre 5ms e 200ms

Figura 9. Comparativo de Desempenho em Busca de Caminho Mais Curto.

7.1. Resultados das Consultas Comparativas de Desempenho

Aqui estão os resultados comparativos de desempenho (tempo de execução) entre o Neo4j e o MySQL para as sete consultas estratégicas.

Q1: Agregação (TOP 10 Conexões)

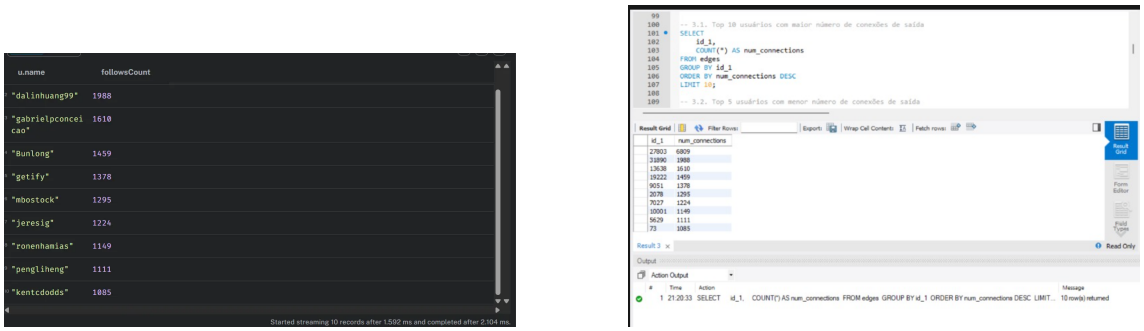

Figura 10. Comparativo de Desempenho em Busca de Agregação (TOP 10 Conexões).

Q2: Agregação (BOTTOM 5 Conexões)

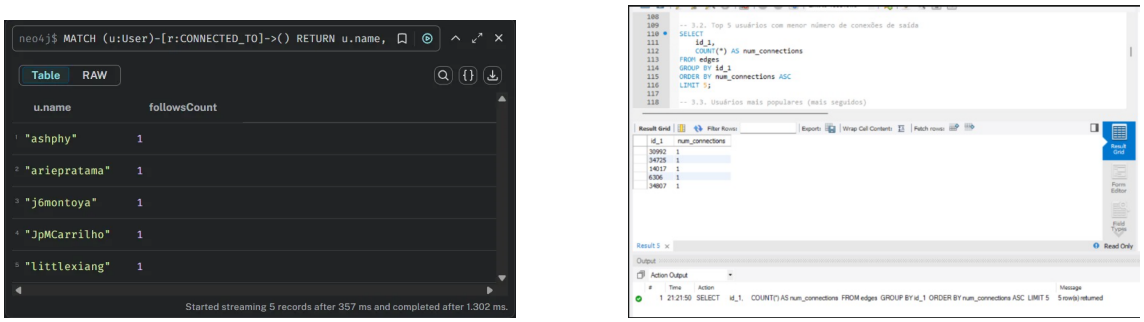

Figura 11. Comparativo de Desempenho em Busca de Agregação (BOTTOM 5 Conexões).

Q3: Buscas por Parte de Texto (CONTAINS)

Q4: Buscas por Padrão Complexo (REGEXP)

Q5: Relação (Vizinhos de 1º Grau - Neo4j: 12 ms)

Q6: Relação (Amigos dos Amigos - Neo4j: 44 ms)

Q7: Relação (Caminho Mais Curto - Neo4j: 0.484 ms)

—

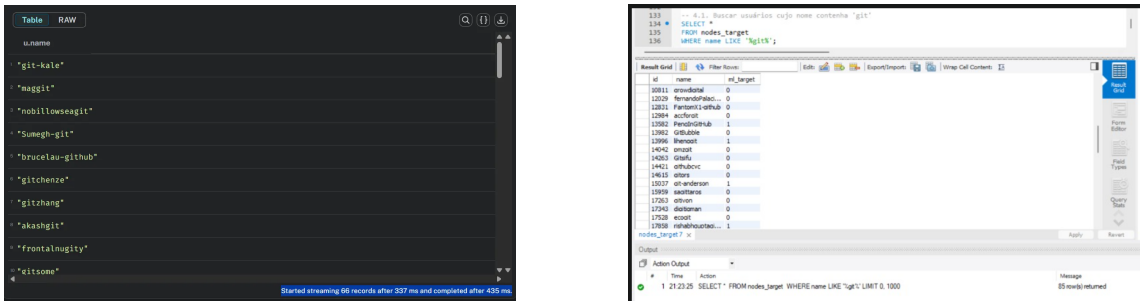

Figura 12. Comparativo de Desempenho em Buscas de Texto (Neo4j: CONTAINS vs MySQL: LIKE).

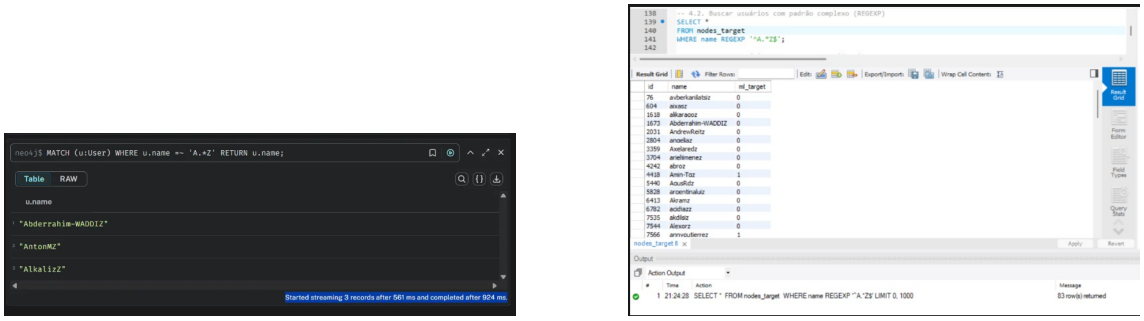

Figura 13. Comparativo de Desempenho em Buscas de Padrão Complexo (Neo4j: =~ vs MySQL: REGEXP).

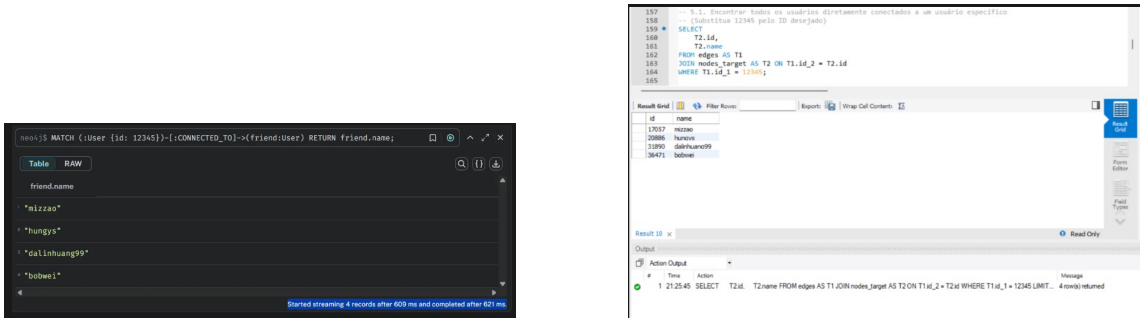

Figura 14. Comparativo de Desempenho em Busca de Relação (1º Grau).

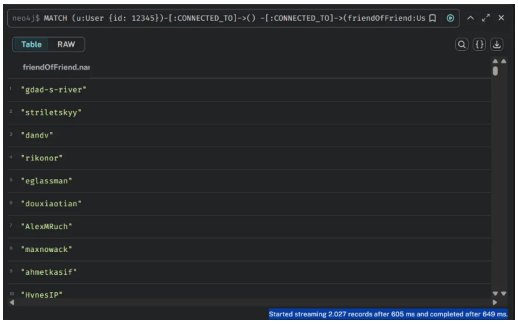

Figura 15. Comparativo de Desempenho em Busca de Relação (2º Grau - Amigos dos Amigos).

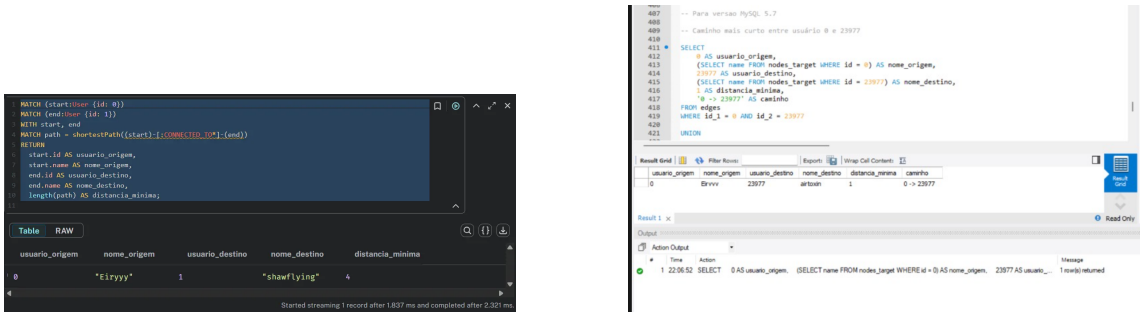

Figura 16. Comparativo de Desempenho em Busca de Caminho Mais Curto (*Shortest Path*).

8. Importação de Dados no Neo4j

A importação do dataset MUSAE GitHub Social Network no Neo4j foi realizada em etapas para garantir a correta indexação e integridade das relações, otimizando o desempenho para futuras consultas.

8.1. Pré-processamento e Constraint

Antes de carregar o arquivo, é fundamental definir uma *constraint* de unicidade no atributo ‘id‘ dos usuários. Isso garante que não haja duplicatas e cria automaticamente um índice B-Tree, essencial para a performance dos comandos ‘MERGE‘.

```
CREATE CONSTRAINT unique_user_id IF NOT EXISTS
FOR (u:User) REQUIRE u.id IS UNIQUE;
```

Listing 1. Cypher: Criação de Constraint para Usuário

8.2. Importação dos Nós (Usuários)

O arquivo `musae_git_target.csv` foi usado para criar os nós da rede, atribuindo-lhes a *label* `:User`.

```
LOAD CSV WITH HEADERS FROM 'file:///musae_git_target.csv' AS row
MERGE (u:User {id: toInteger(row.id)})
SET u.name = row.name,
    u.ml_target = toInteger(row.target);
```

Listing 2. Cypher: Importação dos Nós (musae_git_target.csv)

8.3. Importação das Relações (Arestas)

O arquivo `musae_git_edges.csv` foi utilizado para criar as conexões. O comando ‘MATCH‘ localiza os nós de origem e destino usando o ID e o ‘MERGE‘ estabelece a relação `[:FOLLOWS]` entre eles.

```
LOAD CSV WITH HEADERS FROM 'file:///musae_git_edges.csv' AS row
MATCH (u1:User {id: toInteger(row.id_1)})
MATCH (u2:User {id: toInteger(row.id_2)})
MERGE (u1)-[:FOLLOWS]->(u2);
```

Listing 3. Cypher: Importação das Relações (musae_git_edges.csv)