INSTITUTO FEDERAL DE SÃO PAULO

CAMPUS CAMPOS DO JORDÃO

Estudo da ferramenta Neo4j: funcionalidades e aplicações

Aluno: Luiz Henrique Oliveira de Freitas

Prontuário: CJ3029336

Disciplina: Banco de Dados 2

IFSP-CJO

2025

RESUMO

Bancos de dados não relacionais tem por finalidade o uso para grandes volumes de dados, assim como manipulação de dados que variam com frequência ou sistemas que usam dados desestruturados ou semiestruturados, somando ao fato de não usarem o sistema de tabelas com colunas e linhas isso os torna SGBDs noSQL, uma vez que o clássico SGBD SQL é fundamentado justamente nas relações entre tabelas. SGBDs no SQL podem usar modelos de dados de diferentes tipos, arquivos, chaves e valores, colunas ou até mesmo grafos. Para este estudo o SGBD Neo4j foi escolhido como foco, um sistema baseado em grafos, aqui vamos passar sobre suas principais ferramentas, funcionalidades e aplicações, incluindo a execução de um pequeno projeto de intuito demonstrativo.

PALAVRAS-CHAVE: NoSQL, Neo4j, Grafos;

INTRODUÇÃO

Bancos de dados não estruturados (NoSQL) foram criados para trabalhar com grandes volumes de informação, dados que sofrem variações frequentes, desestruturados ou semiestruturados, podendo usar diferentes modelos de dados, para diferentes tipos de aplicações. Sendo também uma possibilidade a ser considerada em projetos que exijam grande escalabilidade horizontal ou grande desempenho de leitura/ gravação.

SGBDs baseados em documentos como Mongo DB, que podem trabalhar com arquivos JSON ou XML, entre outros, e cada arquivo pode ter sua própria estrutura.

Também é possível encontrar bancos baseados em chave-valor, como no caso do Amazon DynamoDB, que fazem as conexões por meio de pares de chaves e valores correspondentes, um sistema que entrega muita velocidade e simplicidade, usado comumente em sistemas de cache, sessões e configurações.

Já com foco em volume de dados, que podem ser usados em consultas analíticas e coleta de informações, existem os bancos baseados em sistema colunar, como o Apache Cassandra.

E por fim, SGBDs baseados em grafos, como o Neo4j, que tem objetivo de representar e facilitar a consulta de dados altamente interconectados, fazendo o uso de vértices e arestas dos grafos, muito comum em sistemas de recomendações, redes sociais e mapeamento de conexões, o que o torna viável até no rastreio de fraudes.

Para este estudo foi selecionado justamente o SGBD Neo4j, pois é de interesse do autor, uma vez que também se interessa por grafos. Além do mais estudar essa ferramenta abre portas para projetos pessoais futuros, e acredita-se que muitos alunos escolherão MongoDB ou outros por serem mais simples, escolher o Neo4j já é um diferencial por si só, e um investimento para um futuro próximo.

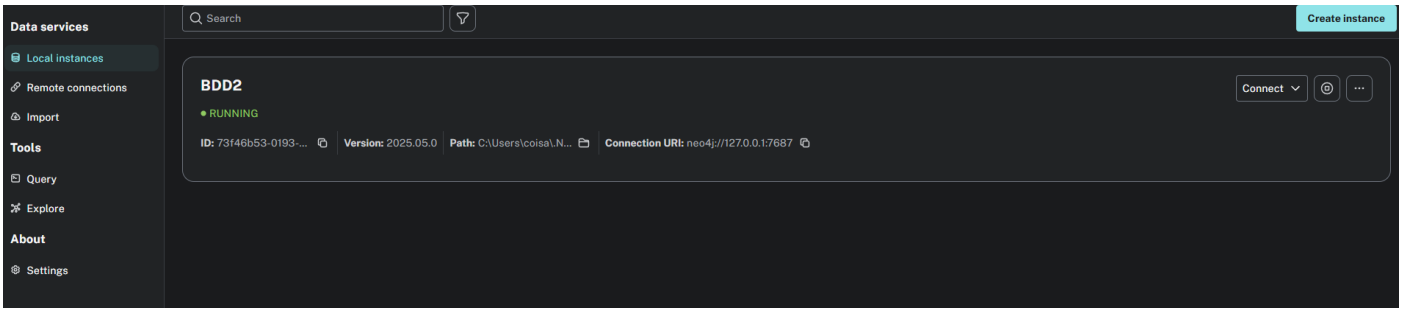
METODOLOGIA

Como informado anteriormente, utiliza-se aqui o *Software* Neo4j como objeto principal de estudo. Para entendimento da ferramenta, após instala-lo e estuda seu uso com recursos da internet, foi desenvolvido um pequeno projeto de finalidade acadêmica de demonstração.

Porém antes de seguirmos ao projeto desenvolvido, vamos abordar o SGBD, sua interface e capacidades de processamento básicas.

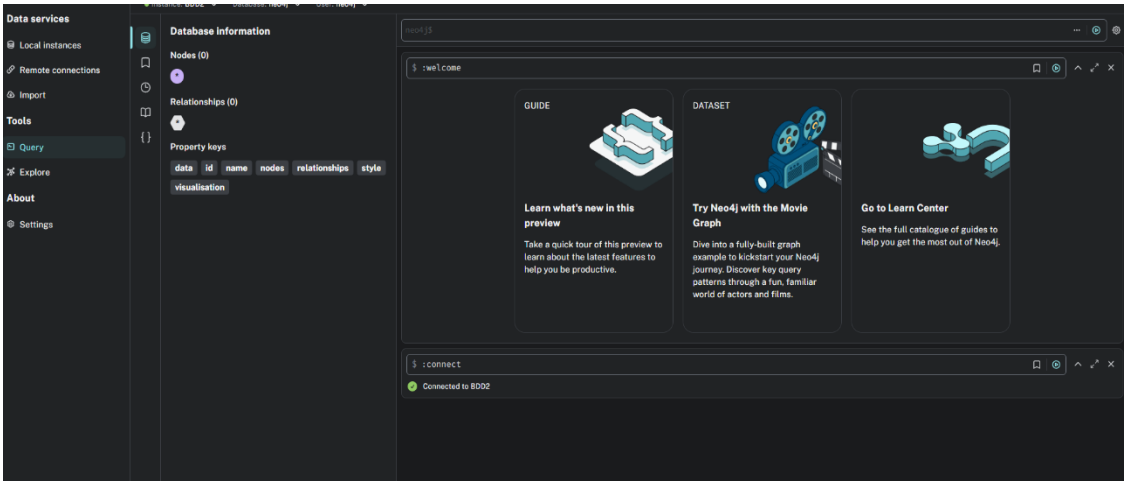
É importante enfatizar que existem cerca de 3 modos de operar o Neo4j, sendo pelo *Browser*, mais acessível e com menos funcionalidades, pelo Neo4j Aura, serviço gerenciado em nuvem e o *Desktop*, utilizado aqui. Além de *APIs* de drivers para integração com Java, Javascript, Python e outros.

Na página inicial podemos gerenciar diferentes bancos de dados no sistema:



Sendo possível também fazer conexões remotamente em outros bancos, assim como importar bancos já construídos, como é possível ver na seção *Data Service* no menu lateral esquerdo.

E na seção *tools* é possível ver as ferramentas Query, onde serão feitos os comandos no BDD, assim como o Explore, para visualização gráfica do BDD como veremos no capítulo seguinte:



Para elaboração do projeto prático, optou-se pela criação de um projeto relacionado a ciclismo, uma vez que esta alinhado com os gostos pessoais do autor.

Foi desenvolvido um banco com dados (nós) de diferentes ciclistas, rotas, pontos de encontro e grupos, e relacionamentos (ligações) entres os diferentes tipos de dados.

O Neo4j usa, diferente do tradicional SQL, a linguagem cypher que serve justamente para trabalhar com dados orientados a grafos. Ao se declarar um nó, usa-se uma sintaxe similar à do exemplo:

```
CREATE (c1:Ciclista {nome: "Luiz", idade: 24, nivel: "intermediário"})
```

Onde:

- CREATE é responsável por criar o nó;
- C1: Ciclista funciona como = nome/apelido do nó : tipo do nó;
- O conteúdo dentro de “{...}” demarca as características do nó;

E para criação de relações seria algo similar a:

```
MATCH (c:Ciclista {nome: "Luiz"}), (r:Rota {nome: "Ciclovia Central"})
CREATE (c)-[:PEDALA_EM]->(r)
```

Onde:

- MATCH declara apelidos temporários a nós existentes, algo que pode ser comparado a o funcionamento de um ponteiro de escopo local;
- No exemplo “c” corresponde ao nó de ciclista onde o nome = luiz, enquanto “r” corresponde a nó de rota onde o nome = Ciclovia Central;
- CREATE torna real a relação direcional de “c” para “r”, :PEDALA_EM;

Em uma pesquisa usa-se novamente o recurso de MATCH:

```
MATCH (c:Ciclista)-[:ENCONTROU_EM]->(:PontoDeEncontro {nome: "Parque das Trilhas"})
RETURN c.nome AS Ciclista
```

Onde:

- Onde c: é uma variável temporária novamente;
- O conteúdo entre “[...]” é a pesquisa;
- E “ -> ” relaciona c: com o que estiver dentro do último (: ...);

RESULTADOS

Já foi abordado antes, sendo assim, aqui será então abordado o desenvolvimento de um Banco de dados formado por dados relacionados a ciclismo. Como a sintaxe do cypher já foi abordada previamente, serão colocados os códigos com simples comentários de desenvolvimento usados para facilitar a compreensão.

Começando pela criação dos nós:

```
// Ciclistas
CREATE (c1:Ciclista {nome: "Luiz", idade: 24, nivel: "intermediário"})
CREATE (c2:Ciclista {nome: "Ana", idade: 29, nivel: "avançado"})
CREATE (c3:Ciclista {nome: "Carlos", idade: 35, nivel: "iniciante"})
CREATE (c4:Ciclista {nome: "Marina", idade: 27, nivel: "intermediário"})

// Rotas
CREATE (r1:Rota {nome: "Trilha da Serra", distancia_km: 25, tipo: "trilha"})
CREATE (r2:Rota {nome: "Ciclovia Central", distancia_km: 10, tipo: "urbana"})
CREATE (r3:Rota {nome: "Estrada Velha", distancia_km: 40, tipo: "estrada"})

// Pontos de Encontro
CREATE (p1:PontoDeEncontro {nome: "Praça Central", bairro: "Centro"})
CREATE (p2:PontoDeEncontro {nome: "Parque das Trilhas", bairro: "Serra Verde"})

// Grupos
CREATE (g1:Grupo {nome: "Pedal Urbano", estilo: "lazer"})
CREATE (g2:Grupo {nome: "Trilheiros Pro", estilo: "competitivo"})
```

Aqui foram criados:

- 4 ciclistas com nome, idade e nível de pedal individual;
- 3 rotas com nome, distancia e tipo cada uma;
- 2 pontos de encontro com nome e bairro cada;
- 2 grupos, com nome e estilo próprios;

Já nas conexões, lembrando que existem inúmeros modos de interconectar dados, um ciclista pode participar de um grupo assim como pode ser amigo de outro ciclista, ou mesmo ser rival de outro ciclista, assim como uma rota pode passar por outra, ou passar por um ponto de encontro. Nada impede que um ciclista participe dos 2 grupos ou de nenhum deles. Veja os exemplos abaixo:

```
// Ciclistas e Rotas
MATCH (c:Ciclista {nome: "Luiz"}), (r:Rota {nome: "Ciclovia Central"})
CREATE (c)-[:PEDALA_EM]->(r)

MATCH (c:Ciclista {nome: "Ana"}), (r:Rota {nome: "Trilha da Serra"})
CREATE (c)-[:PEDALA_EM]->(r)

MATCH (c:Ciclista {nome: "Carlos"}), (r:Rota {nome: "Ciclovia Central"})
CREATE (c)-[:PEDALA_EM]->(r)

MATCH (c:Ciclista {nome: "Marina"}), (r1:Rota {nome: "Trilha da Serra"}), (r2:Rota {nome: "Estrada Velha"})
CREATE (c)-[:PEDALA_EM]->(r1)
CREATE (c)-[:PEDALA_EM]->(r2)
```

```
// Ciclistas e Grupos
MATCH (c:Ciclista {nome: "Luiz"}), (g:Grupo {nome: "Pedal Urbano"})
CREATE (c)-[:MEMBRO_DO]->(g)

MATCH (c:Ciclista {nome: "Ana"}), (g:Grupo {nome: "Trilheiros Pro"})
CREATE (c)-[:MEMBRO_DO]->(g)

MATCH (c:Ciclista {nome: "Marina"}), (g:Grupo {nome: "Trilheiros Pro"})
CREATE (c)-[:MEMBRO_DO]->(g)

// Grupos e Pontos de Encontro
MATCH (g:Grupo {nome: "Pedal Urbano"}), (p:PontoDeEncontro {nome: "Praça Central"})
CREATE (g)-[:USA_COMO_PONTO]->(p)

MATCH (g:Grupo {nome: "Trilheiros Pro"}), (p:PontoDeEncontro {nome: "Parque das Trilhas"})
CREATE (g)-[:USA_COMO_PONTO]->(p)
```

```
// Encontros de ciclistas
MATCH (c:Ciclista {nome: "Luiz"}), (p:PontoDeEncontro {nome: "Praça Central"})
CREATE (c)-[:ENCONTROU_EM]->(p)

MATCH (c:Ciclista {nome: "Carlos"}), (p:PontoDeEncontro {nome: "Praça Central"})
CREATE (c)-[:ENCONTROU_EM]->(p)

MATCH (c:Ciclista {nome: "Ana"}), (p:PontoDeEncontro {nome: "Parque das Trilhas"})
CREATE (c)-[:ENCONTROU_EM]->(p)

MATCH (c:Ciclista {nome: "Marina"}), (p:PontoDeEncontro {nome: "Parque das Trilhas"})
CREATE (c)-[:ENCONTROU_EM]->(p)

// Relação de amizade
MATCH (c1:Ciclista {nome: "Luiz"}), (c2:Ciclista {nome: "Carlos"})
CREATE (c1)-[:AMIGO_DE]->(c2)

MATCH (c1:Ciclista {nome: "Ana"}), (c2:Ciclista {nome: "Marina"})
CREATE (c1)-[:AMIGO_DE]->(c2)
```

Temos 5 tipos de relacionamentos:

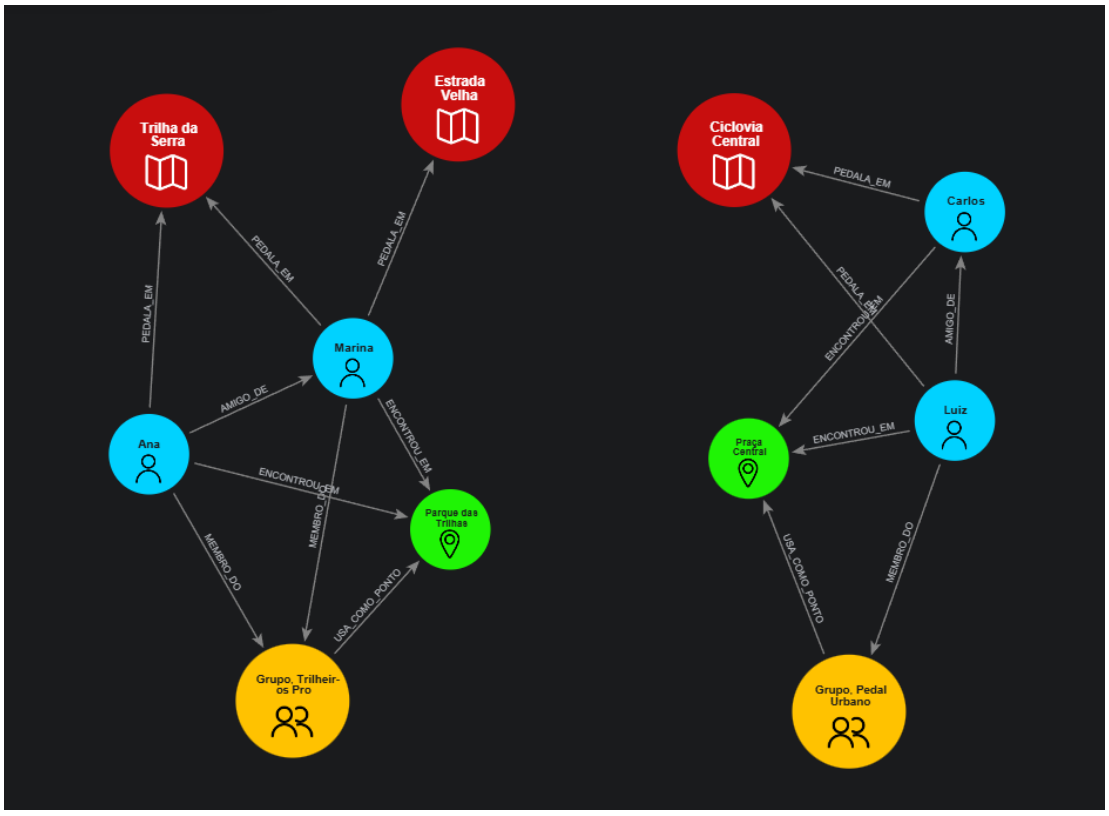
- Ciclistas e rotas;
- Ciclistas e grupos;
- Grupos e Pontos de encontro;
- Encontros de ciclistas;
- Relação de amizade;

**Repare que existe a demarcação vermelha sublinhando os códigos na imagem, isso se da porque é possível criar apenas uma relação por comando. Até pode ser possível criar varias ao mesmo tempo, mas é necessário usar outra estruturação de sintaxe.*

Com nossos dados armazenados, esse grafo permite demonstrar:

- Quais ciclistas pedalam em quais rotas;
- Quais ciclistas fazem parte do mesmo grupo;
- Quem costuma se encontrar em determinados pontos;
- Quem tem mais conexões sociais (via AMIGO_DE);
- Quais grupos usam quais pontos de encontro;
- Conexões indiretas entre ciclistas (ex: por rota, grupo ou amizade);

No momento nós temos um simples Banco de dados construído com o modelo de grafos, veja uma imagem da janela explore do Neo4j com o nosso banco populado desses poucos dados:



A seguir veja 10 pesquisas e seus resultados em nosso BDD no Neo4j:

1 – Mostrar todas as rotas em que “Marina” pedala:

- Código cypher:

```
MATCH (:Ciclista {nome: "Marina"})-[:PEDALA_EM]->(r:Rota)
RETURN r.nome AS Rota, r.distancia_km AS Distância, r.tipo AS Tipo
```

- Resultado da pesquisa:

	Rota	Distância	Tipo
1	"Trilha da Serra"	25	"trilha"
2	"Estrada Velha"	40	"estrada"

2 – Mostrar todos os membros de “Trilheiros Pro”:

- Código cypher:

```
MATCH (c:Ciclista)-[:MEMBRO_DO]->(:Grupo {nome: "Trilheiros Pro"})
RETURN c.nome AS Nome, c.nivel AS Nível
```

- Resultado da pesquisa:

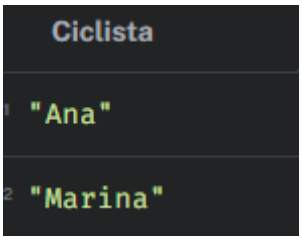
	Nome	Nível
1	"Ana"	"avançado"
2	"Marina"	"intermediário"

3 – Quem se encontra no “Parque das Trilhas”:

- Código cypher:

```
MATCH (c:Ciclista)-[:ENCONTROU_EM]->(:PontoDeEncontro {nome: "Parque das Trilhas"})
RETURN c.nome AS Ciclista
```

- Resultado da pesquisa:



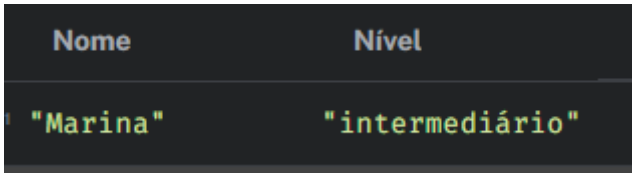
Ciclista
"Ana"
"Marina"

4 – Mostrar todos os amigos de “Ana”:

- Código cypher:

```
MATCH (:Ciclista {nome: "Ana"})-[:AMIGO_DE]->(amigo:Ciclista)
RETURN amigo.nome AS Nome, amigo.nivel AS Nível
```

- Resultado da pesquisa:



Nome	Nível
"Marina"	"intermediário"

5 – Descobrir se ciclistas estão conectados por rotas compartilhadas: (nesse caso verifica quem está conectado a Mariana)

- Código cypher:

```
MATCH (c1:Ciclista {nome: "Marina"})-[:PEDALA_EM]->(r:Rota)<-[:PEDALA_EM]-(c2:Ciclista)
WHERE c2.nome <> "Marina"
RETURN DISTINCT c2.nome AS CiclistaConectado, r.nome AS RotaCompartilhada
```

- Resultado da pesquisa:

CiclistaConectado	RotaCompartilhada
"Ana"	"Trilha da Serra"

6 – Quais ciclistas já pedalaram juntos na mesma rota:

- Código cypher:

```
MATCH (c1:Ciclista)-[:PEDALA_EM]->(r:Rota)<-[:PEDALA_EM]-(c2:Ciclista)
WHERE c1.nome < c2.nome  // evita duplicação tipo A-B e B-A
RETURN DISTINCT c1.nome AS Ciclista1, c2.nome AS Ciclista2, r.nome AS Rota
```

- Resultado da pesquisa:

	Ciclista1	Ciclista2	Rota
1	"Carlos"	"Luiz"	"Ciclovia Central"
2	"Ana"	"Marina"	"Trilha da Serra"

**mostra que Cilista1 e Cliclista2 já peladaram junto em rota x – cada linha é uma resposta.*

7 – Qual ponto de encontro é usado por mais grupos:

- Código cypher:

```
MATCH (:Grupo)-[:USA_COMO_PONTO]->(p:PontoDeEncontro)
RETURN p.nome AS Ponto, COUNT(*) AS TotalDeGrupos
ORDER BY TotalDeGrupos DESC
```

- Resultado da pesquisa:

	Ponto	TotalDeGrupos
1	"Praça Central"	1
2	"Parque das Trilhas"	1

8 – Ciclistas de grupos diferentes que se encontram no mesmo ponto:

- Código cypher:

```
MATCH (c1:Ciclista)-[:ENCONTROU_EM]->(p:PontoDeEncontro)<-[:ENCONTROU_EM]-(c2:Ciclista)
WHERE c1 <> c2
AND NOT (c1)-[:MEMBRO_DO]->()<-[:MEMBRO_DO]-(c2)
RETURN DISTINCT c1.nome AS Ciclista1, c2.nome AS Ciclista2, p.nome AS Ponto
```

- Resultado da pesquisa:

	Ciclista1	Ciclista2	Ponto
1	"Carlos"	"Luiz"	"Praça Central"

**Não entendi exatamente porque Ana e Mariana não aparecem na pesquisa.*

9 – Mostrar grupos com maior diversidade de níveis:

- Código cypher:

```
MATCH (c:Ciclista)-[:MEMBRO_DO]->(g:Grupo)
RETURN g.nome AS Grupo, COLLECT(DISTINCT c.nivel) AS Níveis, COUNT(*) AS TotalMembros
ORDER BY SIZE(COLLECT(DISTINCT c.nivel)) DESC
```

- Resultado da pesquisa:

	Grupo	Níveis	TotalMembros
1	"Trilheiros Pro"	["avançado", "intermediário"]	2
2	"Pedal Urbano"	["intermediário"]	1

10 – Visualizar a rede social de amizades entre ciclistas:

- Código cypher:

```
MATCH (c1:Ciclista)-[:AMIGO_DE]->(c2:Ciclista)
RETURN c1.nome AS Ciclista1, c2.nome AS Ciclista2
```

- Resultado da pesquisa:

	Ciclista1	Ciclista2
1	"Luiz"	"Carlos"
2	"Ana"	"Marina"

**mais uma vez é uma resposta em linha,*

onde cada linha mostra as relações de amizade

CONCLUSÃO

Neo4j é uma ferramenta muito interessante, não foi abordado neste arquivo, mas é possível utilizar operadores lógicos e relacionais nas pesquisas como visto em alguns dos exemplos, além de outros recursos de busca, como modos de ordenação e demais comandos. Como o foco desta obra não é destrinchar o uso da ferramenta e seus comandos, a disponibilidade de tais funcionalidade foi deixada implícita.

Assim como varias pequenas funcionalidades legais do software, que tem um sistema de ordenar pesquisa por coluna desejada, e diversos tipos de customização para as visualizações gráficas dos grafos, como é mostrado superficialmente na página 8.

Esse é um SGBD de fácil entendimento, aqui não nos aprofundamos em suas capacidades tanto quanto necessário para trabalhar com essa ferramenta, apenas passamos por suas funcionalidades básicas e entendemos um pouco do que ela pode fazer e como funciona um SGBD relacionado a grafos. Fazer essa pesquisa de desenvolvimento foi um projeto interessante e muito enriquecedor, será de grande ajuda para aplicar bancos como esse em projetos futuros.