Banco de Dados Relacionais e Não Relacionais

Prof. Henrique Batista da Silva

Banco de dados de Grafos

Álbum de músicas

Considere uma aplicação que contém o perfil de **músicos**, **bandas**, **álbuns** e **músicas**.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

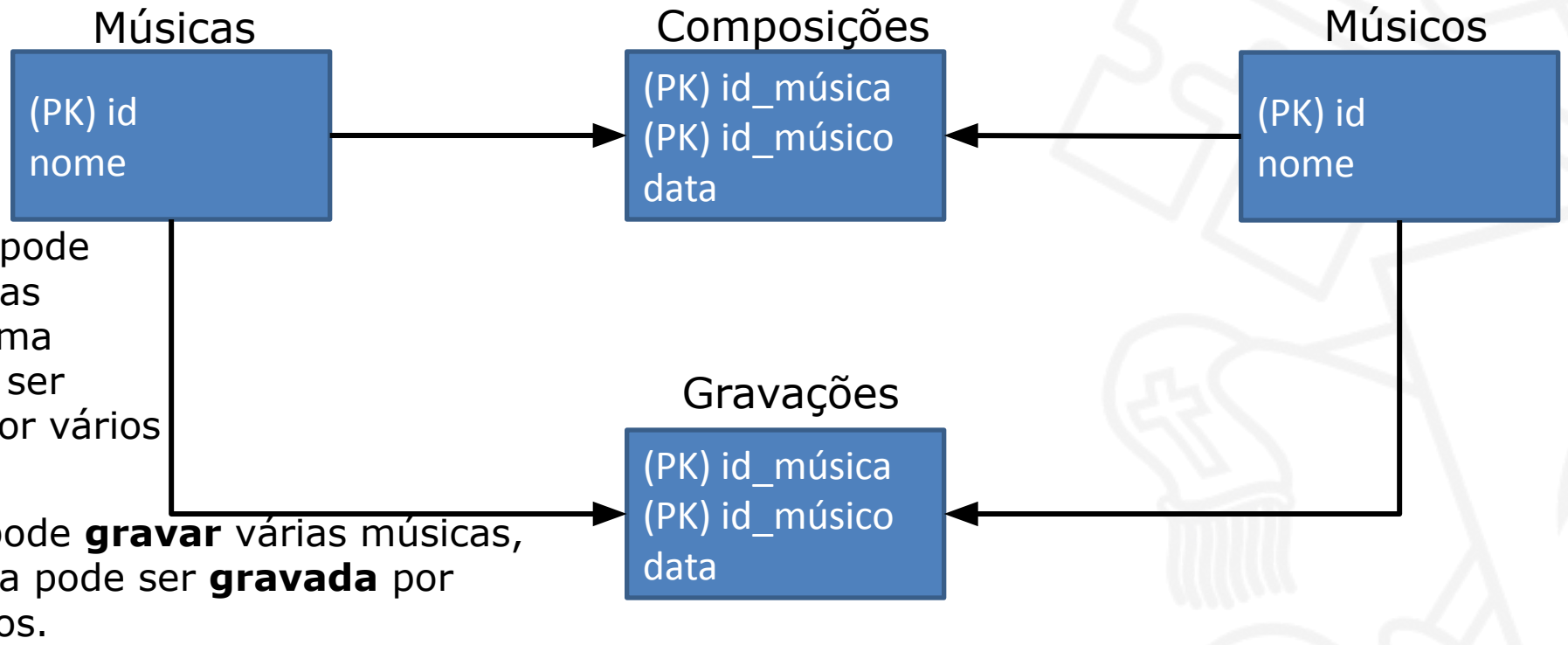
Relacionamento entre artistas

Suponha agora que seja necessário o cadastro de músicos e bandas para mapear as parcerias formadas.

Podemos armazenar músicas e músicos que gravam (e compõe) as músicas e encontrar relações entre músicos.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Relacionamento entre artistas



Um músicos pode **compor** várias músicas, e uma música pode ser **composta** por vários músicos.

Um músico pode **gravar** várias músicas, e uma música pode ser **gravada** por vários músicos.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Relacionamento entre artistas

Para uma simples consulta, suponha que queiramos retornar os compositores e suas músicas:

```
SELECT mo.nome as compositor, ma.nome as musica
FROM composicoes c
JOIN musicas ma on ma.id = c.musica_id
JOIN musicos mo on mo.id = c.musico_id
```

compositor	musica
Bob Dylan	All Along the Watchtower
Bob Dylan	One More Cup of Coffee
Desmond Child	Crazy
Desmond Child	Livin' on a Prayer

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Relacionamento entre artistas

Suponha agora que queiramos retornar a

quantidade de músicas que um músico

gravou do mesmo compositor:

interprete	compositor	Total_músicas
George Harrison	Bob Dylan	1
Jimi Hendrix	Bob Dylan	1
Steve Tyler	Desmond Child	2
Jon Bon Jovi	Desmond Child	1

```
SELECT mo.nome as interprete,
com.nome as compositor,
COUNT(com.id) as total_musicas
FROM musicos mo
JOIN gravacoes g ON mo.id = g.musico_id
JOIN musicas ma ON ma.id = g.musica_id
JOIN composicoes c ON ma.id = c.musica_id
JOIN musicos com ON com.id = c.musico_id
WHERE com.id <> mo.id
GROUP BY com.id, mo.id
ORDER BY mo.nome, com.nome;
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Relacionamento entre artistas

Estes tipos de consultas podem ser utilizados em sistema como *e-commerce* para descobrir, por exemplo, "*Quem comprou este produto também comprou*".

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Relacionamento entre artistas

EM SGBDR, *joins* podem deixar as consultas muito lentas.

Iremos utilizar outro tipo de banco, o Neo4j, para modelar este problema.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

O que é um Grafo?

O que é um Grafo?

O grafo é uma estrutura de dados composta por um conjunto de vértices e um conjuntos de arestas.

As arestas podem ou não serem direcionadas.

O que é um Grafo?

É possível representar diversas situações do mundo real utilizando a teoria de grafos. Por exemplo, podemos representar um centro urbano, em que cada esquina é um vértice v e sempre que houver uma rua ligando uma esquina v_1 a uma esquina v_2, criamos uma aresta e entre os vértices v_1 e v_2

O que é um Grafo?

Observe que neste exemplo, podemos dizer que uma aresta que liga os vértices v_1 e v_2 será uma aresta direcionada, se a rua que liga as esquinas v_1 e v_2 for de mão única.

Da mesma forma, podemos dizer que a resta entre os vértices v_1 e v_2 não será direcionada se a rua for de mão dupla

O que é um Grafo?

Ainda no nosso exemplo, podemos adicionar pesos em nossas arestas, em que o peso de uma aresta e ligando v_1 e v_2 será o valor da distância em metros entre as esquinas v_1 e v_2.

O que é um Grafo?

Da mesma forma, podemos representar a distância entre cidades. Sendo os vértices as cidades e as arestas as estradas que ligam as cidades.

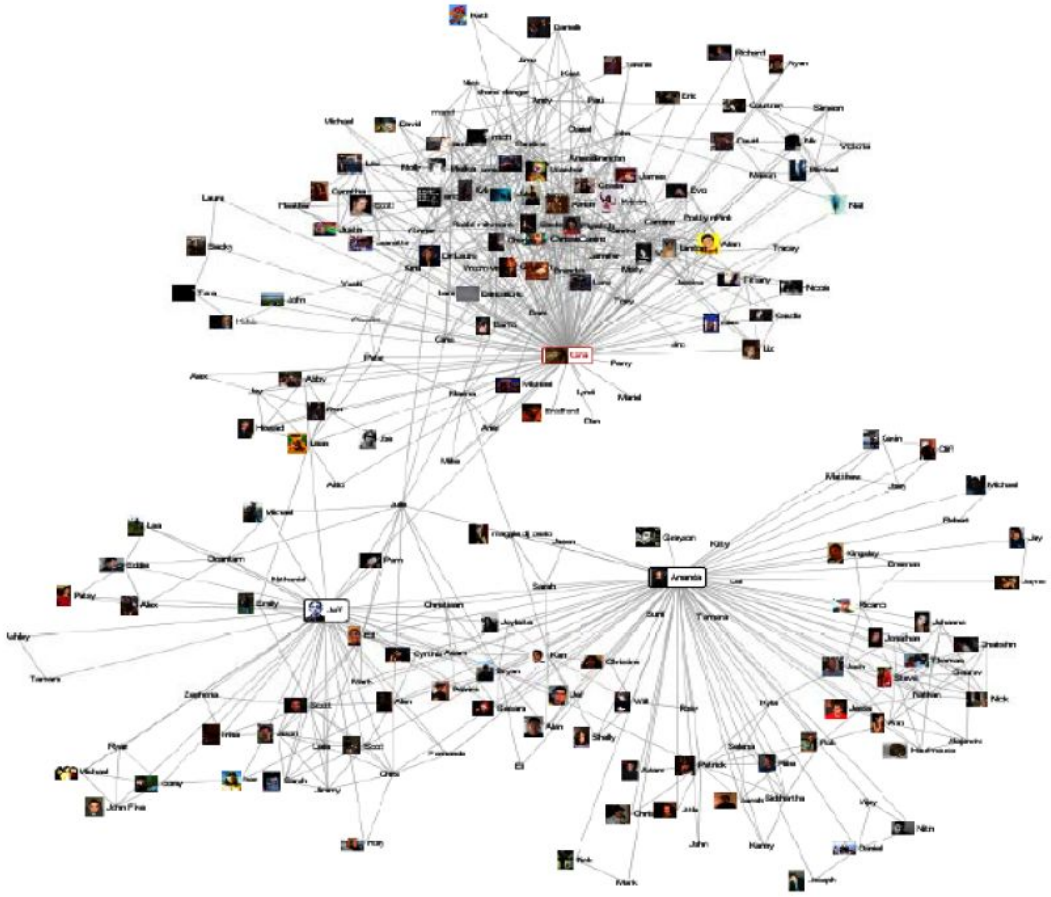
Haverá uma aresta entre dois vértices se houver uma estrada que liga duas cidades diretamente.

O que é um Grafo?

Da mesma forma, podemos representar a distância entre cidades. Sendo os vértices as cidades e as arestas as estradas que ligam as cidades.

Haverá uma aresta entre dois vértices se houver uma estrada que liga duas cidades diretamente.

Rede Social



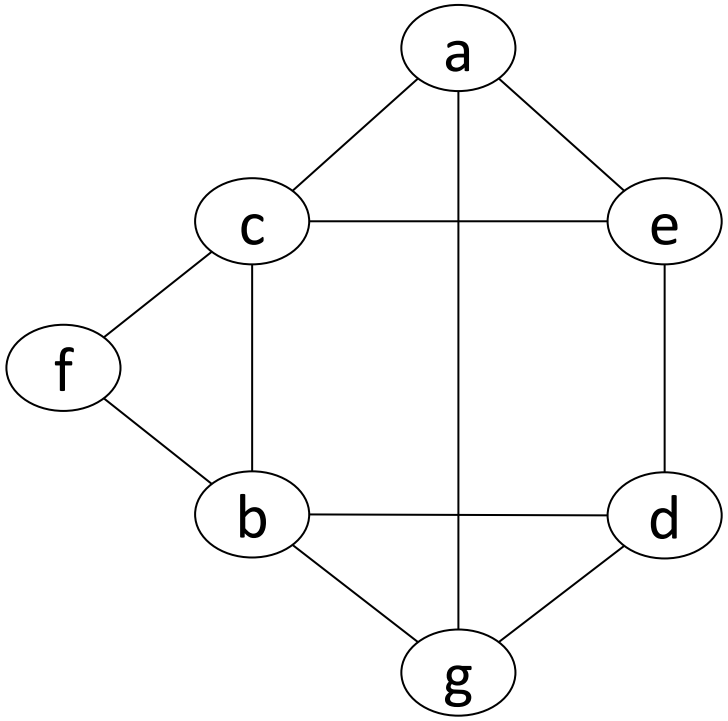
Fonte: "**Vizster: Visualizing Online Social Networks**." Jeffrey Heer and danah boyd. *IEEE Symposium on Information Visualization* (InfoViz 2005).

O que é um Grafo?

Em uma rede social, um grafo pode ser utilizado para modelar o relacionamento entre as pessoas. Em que, cada pessoa será representada por um vértices e haverá uma aresta entre dois vértices se as duas pessoas forem “amigas” na rede.

A aresta será direcionada se uma pessoas “segue” a outra

O que é um Grafo?



Um grafo modelado para representar amizades em uma rede social.

O que é um Grafo?

O grafo é uma estrutura de dados que contem diversos algoritmos para busca, caminho mais curto, entre outros conceitos.

A teoria de grafos possui diversas aplicações na computação e em outras áreas.

Banco de dados Neo4j

Banco de dados de Grafo

O foco é em como os objetos se relacionam.

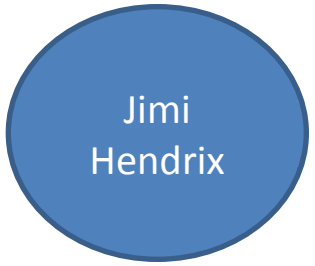
Conceitos de **nodos** (nós) e **arestas**.

Entidades são representados por **nodos** e relacionamentos são representados por **arestas** (que podem conter propriedades).

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

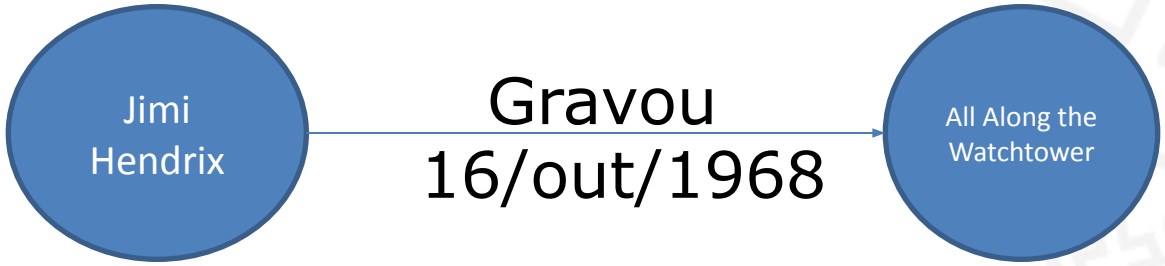
Nodo do grafo representando um artista no banco.



Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

Nodo do grafo representando um artista no banco.



O relacionamento acima (aresta) representa a informação de que "Jimi Hendrix" gravou a música "All Along the Watchtower" em "16/out/1968"

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

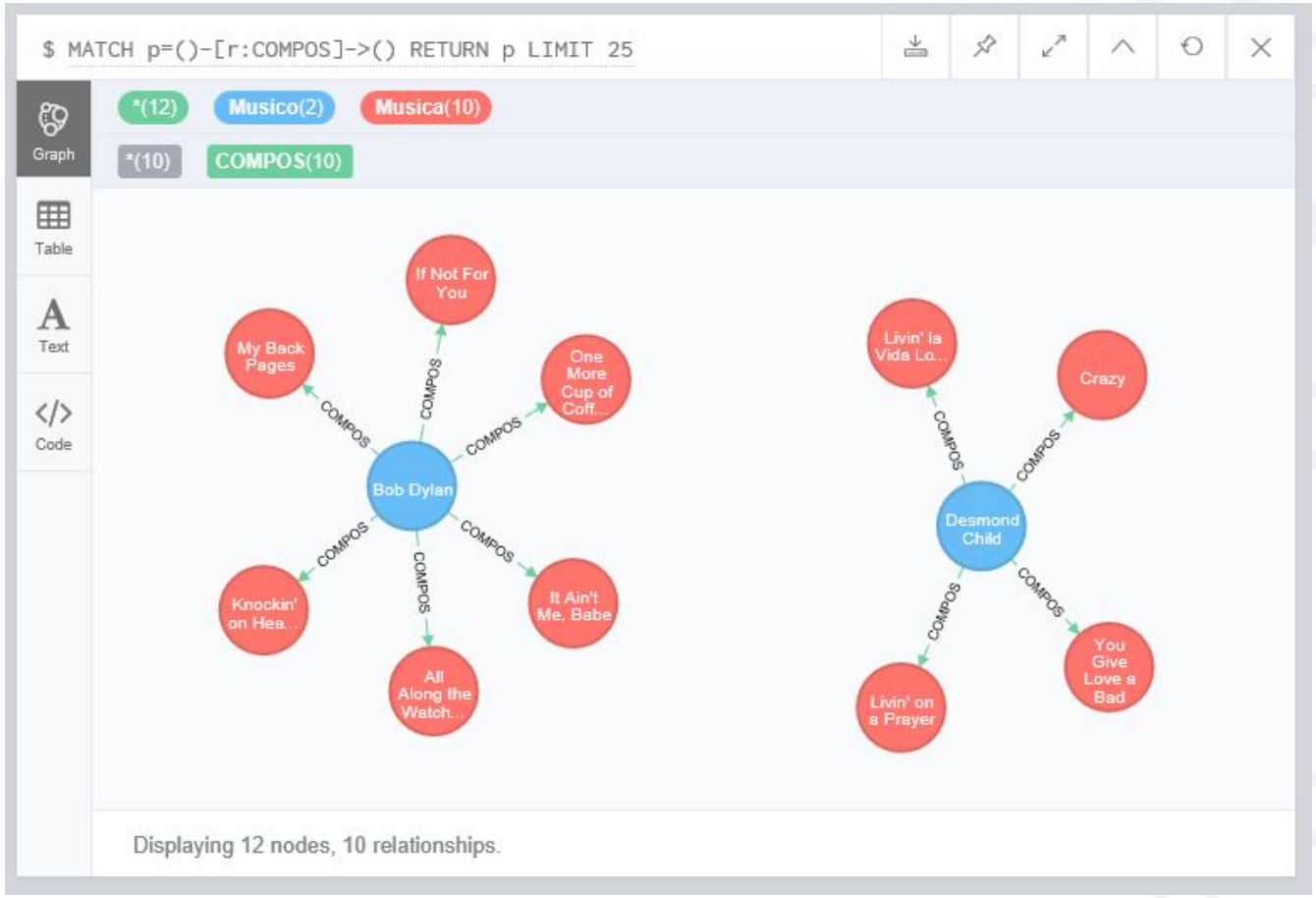
Banco de dados de Grafo

A ideia do banco de dados de Grafos é abstrair toda a complexidade de seus algoritmos e prover ferramenta para explorar as suas características.

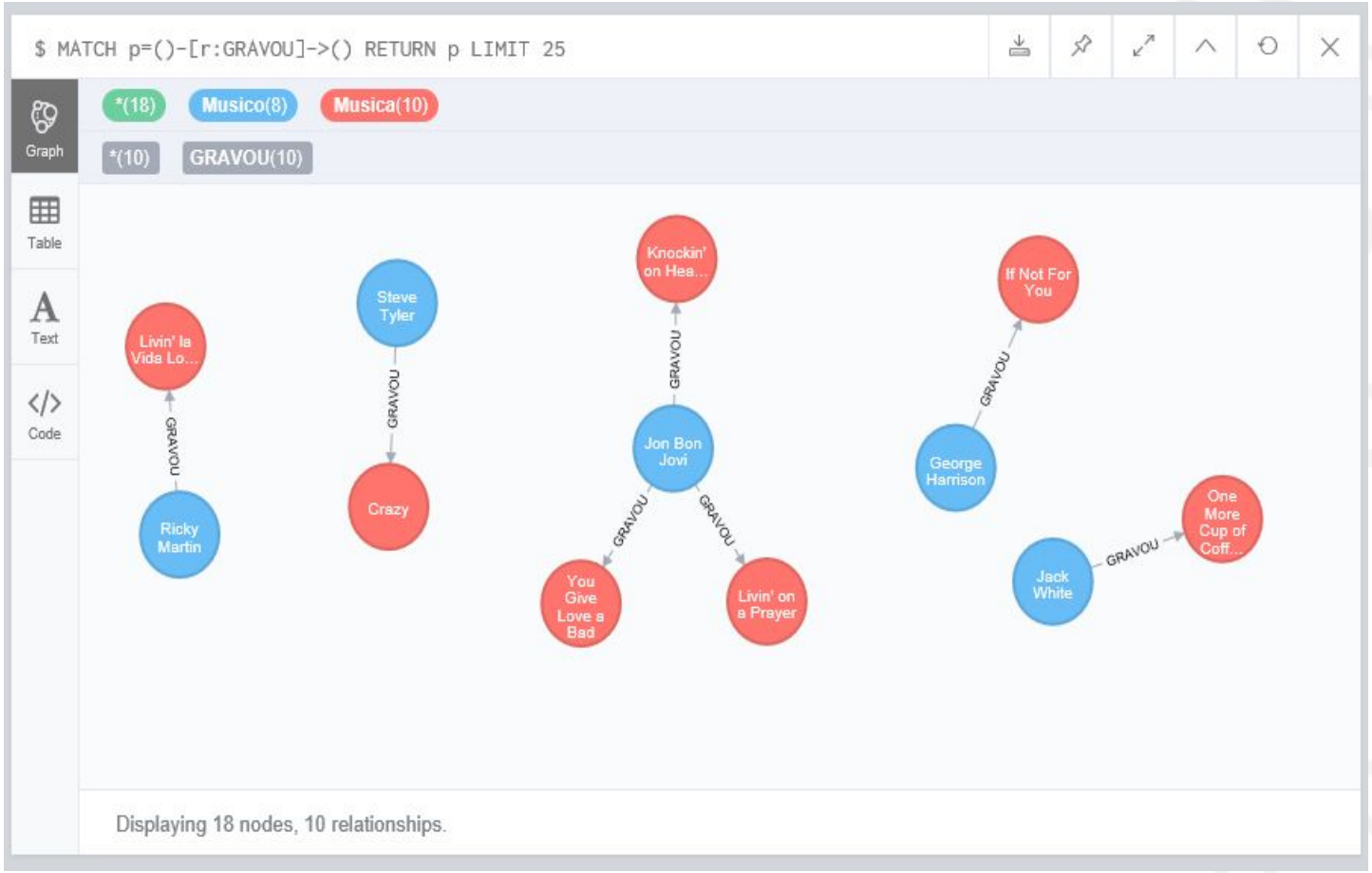
Banco de dados de Grafo

O banco que iremos utilizar, o mais famosos baseado em grafos, chama-se Neo4j.

Neo4j


$ MATCH p=()-[r:COMPOS]->() RETURN p LIMIT 25
Graph
Table
Text
Code
*(12)
Musico(2)
Musica(10)
*(10)
COMPOS(10)
If Not For You
My Back Pages
One More Cup of Coff...
Bob Dylan
Knockin' on Hea...
It Ain't Me, Babe
All Along the Watch...
Livin' la Vida Lo...
Crazy
Desmond Child
Livin' on a Prayer
You Give Love a Bad
COMPOS
Displaying 12 nodes, 10 relationships.

Neo4j


$ MATCH p=()-[r:GRAVOU]->() RETURN p LIMIT 25
Graph
Table
Text
Code
*(18)
Musico(8)
Musica(10)
*(10)
GRAVOU(10)
Livin' la Vida Lo...
Ricky Martin
GRAVOU
Steve Tyler
Crazy
Knockin' on Hea...
Jon Bon Jovi
You Give Love a Bad
Livin' on a Prayer
If Not For You
George Harrison
Jack White
One More Cup of Coff...
Displaying 18 nodes, 10 relationships.

Banco de dados de Grafo

Neo4j Community Edition (Windows 64 bit):

https://neo4j.com/download/community-edition/

Acessando o Neo4j Online

Inserindo nós e arestas no grafo

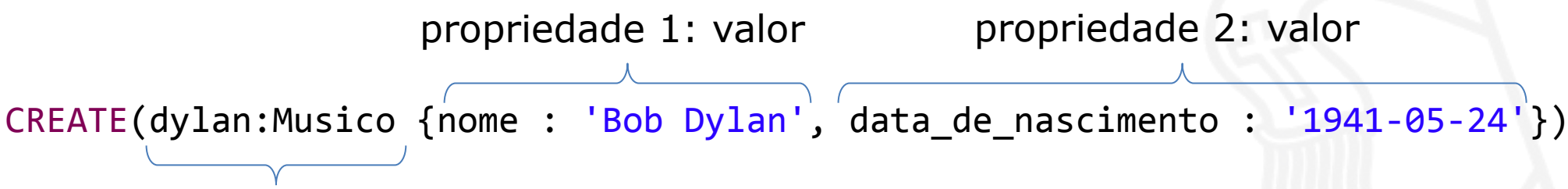
Banco de dados de Grafo

Só há dois tipos de dados no Neo4J: o nó (semelhante ao documento no MongoDB) e o relacionamento (aresta).

Um nós pode ter um tipo e possuir diversos atributos.

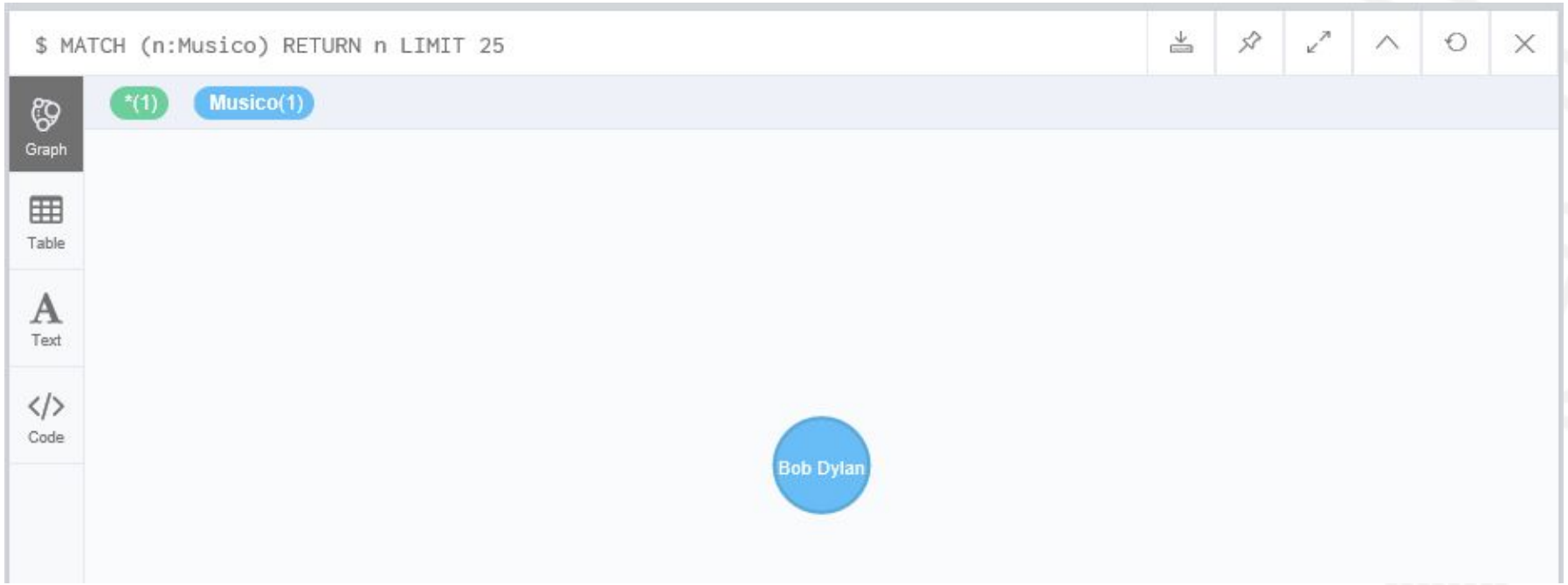
Inserindo um novo nó

O primeiro nó que iremos criar será do tipo músico com nome e dataNascimento. Vamos criar um nó que representa o músico Bob Dylan. Execute o seguinte comando na interface:



Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo



Único nó inserido até o momento no banco de dados

Banco de dados de Grafo

Agora precisamos criar um nó para uma das músicas do artista para então definirmos um relacionamento.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

Um relacionamento pode possuir **tipo** e **propriedades**

A **direção** do relacionamento também é uma característica importante

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

Antes vamos criar dois nós: um músico e uma música (execute e veja o resultado no console).

propriedade 1: valor

```
CREATE(hendrix:Musico {nome : "Jimi Hendrix"})
```

Nome da variável : tipo do nó

```
CREATE(al_along:Musica {nome : "All Along the Watchtower"})
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

Para ciar um relacionamento precisamos definir um **tipo** e uma **direção**.

O relacionamento deve ser do tipo “hendrix” **gravou** “all along”

O relacionamento é do tipo **gravou** e de “hendrix” para “all along”

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo

Execute abaixo o comando para criar o relacionamento no banco. Observe que primeiro precisamos definir as variáveis.

```
MATCH (hendrix:Musico{nome:"Jimi Hendrix"}),(al_along:Musica{nome:"All Along the Watchtower"})
CREATE (hendrix)-[r:GRAVOU]->(al_along)
```

Especifica a estrutura do grafo pesquisado

Nó 1

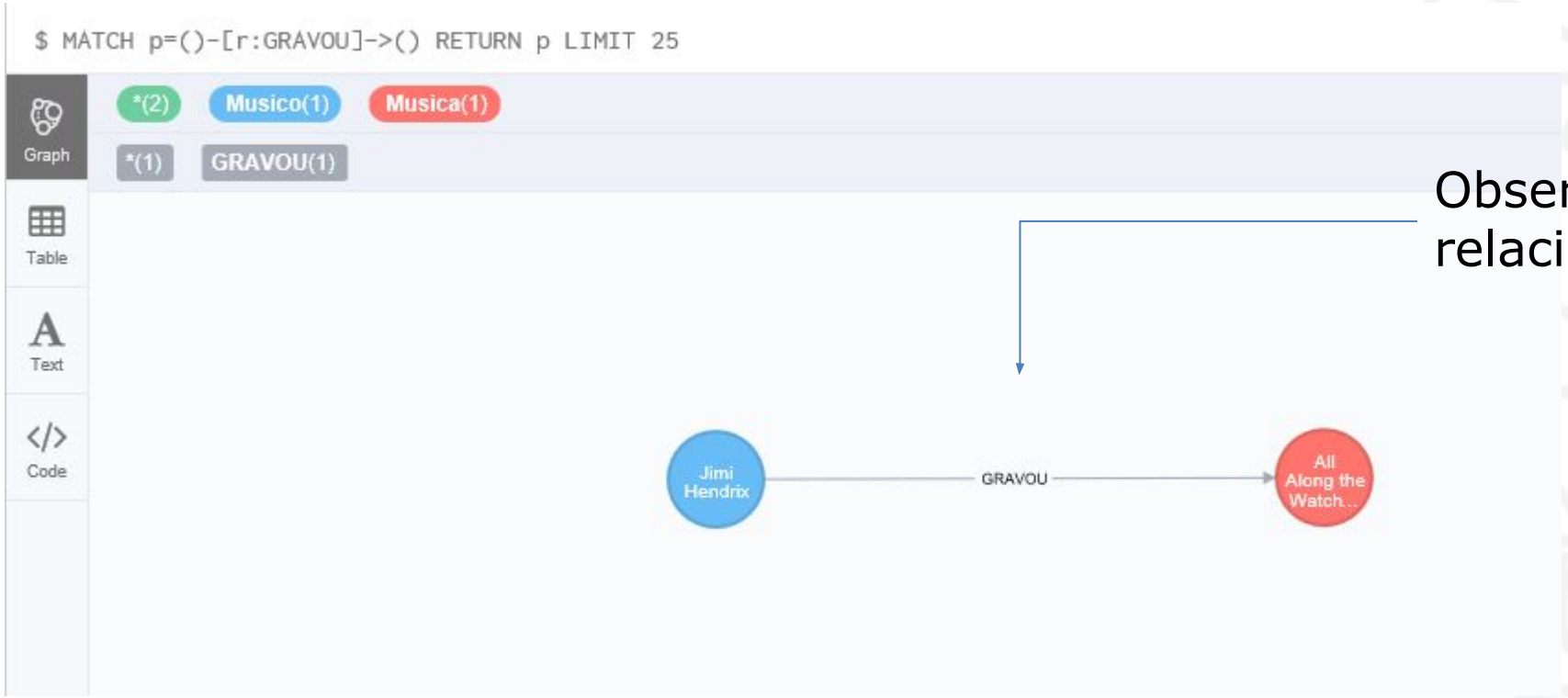
Tipo do relacionamento

Direção do relacionamento

Nó 2

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Banco de dados de Grafo



Observe a direção e o tipo do relacionamento

Consultas no Grafo

Realizando consultas

- Usamos anteriormente o comando MATCH, agora iremos entender melhor o seu funcionamento.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

A linguagem padrão no banco de dados é a **Cypher**, utilizada para realizar consultas.

Usamos a palavra MATCH para definir a estrutura do grafo pesquisado.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

A palavra RETURN faz parte da consulta e é usada para definir o que a consulta irá retornar.

Analogia com banco relacional

RETURN	→	SELECT
MATCH	→	FROM

Realizando consultas

Para listar o nome de todos os artistas no banco de dados, usando cypher.

```
MATCH
(m:Musico)
RETURN m.nome
```



```
FROM
musicos m
```

Realizando consultas

Para listar o nome de todos os artistas no banco de dados, usando cypher.

```
MATCH (m:Musico)
RETURN m.nome
```

O MATCH encontra todos os nós do tipo “Musico”. O resultado é atribuído para a variável “m”

RETURN projeta o atributo “nome” dos nós de “m”

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

Para listar o nome de todos os artistas no banco de dados, usando cypher.

```
MATCH (m:Musico)
RETURN m
```

Retorna nós do banco de dados.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

Para listar o nome de todos os artistas no banco de dados, usando cypher.

```
MATCH (m)
RETURN m
```

Retorna todo o grafo.

Realizando consultas

A linguagem cypher também possui a cláusula WHERE, como em SQL, para aplicar filtros nas consultas.

```
MATCH (m:Musico)
WHERE m.nome = 'Bob Dylan'
RETURN m
```

Entre todos os músicos do banco (cláusula MATCH) filtrar o músico com nome 'Bob Dylan' (cláusula WHERE)

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

Podemos filtrar dentro da cláusula MATCH, informando o atributo e o seu valor.

```
MATCH (m:Musico { nome : 'Bob Dylan' })
RETURN m
```

Produz o mesmo resultado da consulta anterior

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

Crie dois relacionamentos entre o artista Bob Dylan e a música “All Along the Watchtower”. Um do tipo **GRAVOU** e outro do tipo **COMPOS**

Utilize a cláusula MATCH para especificar as variáveis

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas

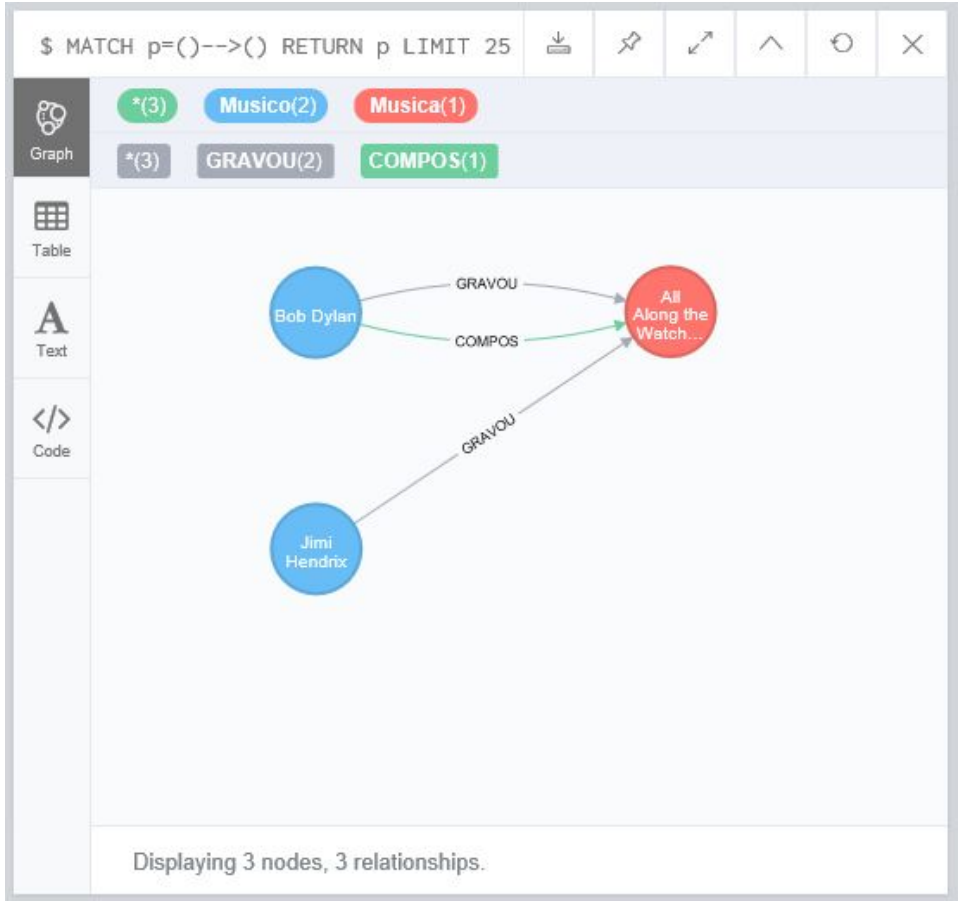
Crie dois relacionamentos entre o artista Bob Dylan e a música “All Along the Watchtower”.

Um do tipo **GRAVOU** e outro do tipo **COMPOS**

```
MATCH (bob:Musico { nome : 'Bob Dylan' }),
(al_along:Musica { nome : 'All Along the Watchtower'})
CREATE (bob)-[r:GRAVOU]->(al_along)
CREATE (bob)-[s:COMPOS]->(al_along)
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Realizando consultas



Observe os dois relacionamentos criados entre “Bob Dylan” e “All Along the Watchtower”

Realizando consultas

A cláusula MATCH também pode ser usada para especificar relacionamentos.

Podemos, por exemplo, buscar pela estrutura “Musico relacionado com MUSICA”.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1)-[]-()
Return n1
```

Retorna os relacionamentos sem especificar a direção

```
{"nome":"All Along the Watchtower"}
{"nome":"Bob Dylan","data_de_nascimento":"1941-05-24"}
{"nome":"Bob Dylan","data_de_nascimento":"1941-05-24"}
{"nome":"Jimi Hendrix"}
{"nome":"All Along the Watchtower"}
{"nome":"All Along the Watchtower"}
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1)-[]->()
Return n1
```

Retorna os relacionamentos especificando a direção. Somente os nós com grau de saída.

```
{"nome":"Bob Dylan","data_de_nascimento":"1941-05-24"}
{"nome":"Bob Dylan","data_de_nascimento":"1941-05-24"}
{"nome":"Jimi Hendrix"}
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1)<-[]-()
Return n1
```

```
{"nome":"All Along the Watchtower"}
{"nome":"All Along the Watchtower"}
{"nome":"All Along the Watchtower"}
```

Retorna os relacionamentos especificando a direção. Somente os nós com grau de chegada.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1)<-[]-()
Return count (n1)
```



```
"count (n1)"

3
```

Retorna a quantidade de arestas de chegada no grafo. Somente os nós com grau de chegada.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1:Musico)-[r]-(n2:Musica)
Return n1, type(r), n2
```

Retorna todos os relacionamentos do tipo “Musico” para “Musica”, especificando os tipos do relacionamento

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Exibindo todos os relacionamentos

Podemos executar uma consulta para exibir todos os relacionamentos do banco

```
MATCH (n1:Musico)-[r:GRAVOU]->(n2:Musica)
Return n1, type(r), n2
```

Retorna todos os relacionamentos do tipo “Musico” GRAVOU “Musica”, filtrando pelo tipo do relacionamento

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Excluir e editar

Editando nós

Como em um banco relacional, no Neo4j também podemos editar e apagar nós e relacionamentos.

Para fazer uma atualização em um nó, precisamos incluir a cláusula SET junto com a cláusula MATCH, especificando o que queremos alterar.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Editando nós

Adicionando o campo data_de_nascimento para o músico “Jimi Hendrix”.

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })
SET hendrix.data_de_nascimento = '1942-11-27'
RETURN hendrix
```

Primeiro buscamos pelo músico com o nome ‘Jimi Hendrix’ e depois adicionamos o novo atributo.

Se o atributo já existir, ele altera o valor do atributo existente. Podemos assim atribuir novos campos ou atualizar seus valores da mesma forma

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Editando nós

Removendo o campo data_de_nascimento para o músico “Jimi Hendrix”.

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })
SET hendrix.data_de_nascimento = null
RETURN hendrix
```

Usamos null para remover um atributo do nó.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Removendo nós

Excluindo o nó “Jimi Hendrix”.

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })
DELETE hendrix
```

Usamos DELETE para remover o nós do banco de dados

Mas antes precisamos excluir os relacionamentos em que o nós está envolvido

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Removendo nós

Primeiro vamos pesquisar todos os tipos de relacionamentos em que ‘Jimi Hendrix’ esteja envolvido.

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })-[r]-()
RETURN hendrix, type(r)
```

Observe que não estamos especificando a direção do relacionamento.

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Removendo nós

Agora podemos remover o relacionamento e o nó logo em seguida.

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })-[r]-()
DELETE r
```

Remove o relacionamento

```
MATCH (hendrix:Musico { nome : 'Jimi Hendrix' })
DELETE hendrix
```

Remove o nó

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Removendo nós

Para apagar todo o banco, independente de ter ou não relacionamentos, usamos a cláusula OPTIONAL.

```
MATCH (n)
OPTIONAL MATCH (n)-[rel]-()
DELETE rel, n
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Importando de Arquivos CSV

Lendo de arquivos csv

Podemos também criar arquivos do tipo csv para importar grande quantidade de dados para o nosso banco.

Vamos aprender como fazer a importação dos arquivos para popular o banco automaticamente.

Referências: Pânez, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Antes, vamos conhecer mais uma cláusula do Cypher, a palavra MERGE.

Ao importar os dados de artistas e músicas, em muitos casos teremos nomes repetidos de um ou outro (um artista grava várias músicas e uma música pode ser gravada por vários artistas)

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Para evitar a inserção de itens duplicados usaremos o MERGE.

```
MERGE (n1:Musico {nome: "Bob Dylan"})
MERGE (n2:Musico {nome: "Bob Dylan"})
```

O primeiro MERGE será como um CREATE, ele irá criar o nó “Bob Dylan”. O segundo MERGE, como o nó já existe, ele apenas ignora e não cria um nó duplicado.

O mesmo se aplica aos relacionamentos

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Baixes os dois arquivos na pasta da aula (composicoes.csv e gravações.csv) e copie e cole na seguinte pasta de seu computador (você precisará criar a pasta)

C:\Users\SeuNome\Documents\Neo4j\default.graphdb\import\

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Vamos utilizar o comando LOAD para carregar o arquivo e vamos apenas imprimir o seu conteúdo (veja o resultado na tela). O LOAD adiciona cada linha do arquivo para a variável "linha".

```
LOAD CSV WITH HEADERS
FROM "file:///composicoes.csv"
AS linha
RETURN linha
```

O diretório "file" representa o diretório criado anteriormente

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Para efetivamente criar os nós, vamos utilizar o comando MERGE.

```
LOAD CSV WITH HEADERS
FROM "file:///composicoes.csv"
AS linha
MERGE (compositor:Musico {nome: linha.compositor})
MERGE (musica:Musica {nome: linha.musica})
MERGE (compositor)-[:COMPOS]->(musica)
```

Observe que o LOAD carrega todas as linhas para a variável linha.
Depois usamos o MERGE para criar os nós do tipo "**Musica**" e "**Musico**".
Também utilizamos o MERGE para criar os relacionamentos direcionados do tipo "**Musico**" "**COMPOS**" "**Musica**"

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Lendo de arquivos csv

Agora carregamos o arquivo de gravações.

```
LOAD CSV WITH HEADERS
FROM "file:///gravacoes.csv"
AS linha
MERGE (interprete:Musico {nome: linha.interprete})
MERGE (musica:Musica {nome: linha.musica})
MERGE (interprete)-[:GRAVOU]->(musica)
```

Observe que se os músicos já existirem, nada será criado.

Observe que o LOAD carrega todas as linhas para a variável linha.
Depois usamos o MERGE para criar os nós do tipo "**Musica**" e "**Musico**".
Também utilizamos o MERGE para criar os relacionamentos direcionados do tipo "**Musico**" "**COMPOS**" "**Musica**"

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Fazendo algumas consultas

Retorne os músicos que gravaram alguma música:

```
MATCH (i:Musico)-[g:GRAVOU]->(m:Musica)
return i, m
```

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

Fazendo algumas consultas

Retorne os músicos que gravaram (interprete) alguma música e músicos que compôs (compositor) alguma música.

```
MATCH (interprete:Musico)-[gravou:GRAVOU]->(musica:Musica)
MATCH (compositor:Musico)-[compos:COMPOS]->(musica:Musica)
RETURN interprete, musica, compositor
```

Observe que a mesma variável deve ser usada aqui

Referências: Paniz, David. NoSQL: Como armazenar os dados de uma aplicação moderna. Casa do Código, 2017.

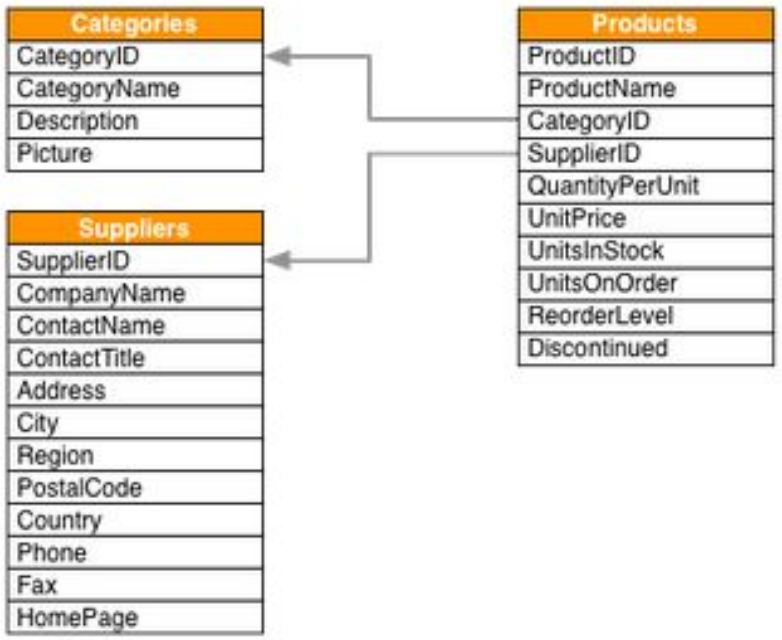
Para saber mais

Acesse o site oficial do Neo4j

https://neo4j.com/developer/guide-import-csv/

Desafio 1: Importando um banco Relacional

Trabalhando com o banco Northwind



A Northwind vende produtos em algumas categorias, fornecidas por fornecedores. Vamos carregar as tabelas do catálogo de produtos.

Trabalhando com o banco Northwind

Execute o comando abaixo para carregar os produtos do arquivo products.csv

```
LOAD CSV WITH HEADERS FROM "file:///products.csv" AS row
CREATE (n:Product)
SET n = row,
n.unitPrice = toFloat(row.unitPrice),
n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder =
toInteger(row.unitsOnOrder),
n.reorderLevel = toInteger(row.reorderLevel), n.discontinued =
(row.discontinued <> "0")
```

Trabalhando com o banco Northwind

Execute o comando abaixo para carregar os produtos do arquivo categories.csv

```
LOAD CSV WITH HEADERS FROM
"file:///categories.csv" AS row
CREATE (n:Category)
SET n = row
```

Trabalhando com o banco Northwind

Execute o comando abaixo para carregar os produtos do arquivo suppliers.csv

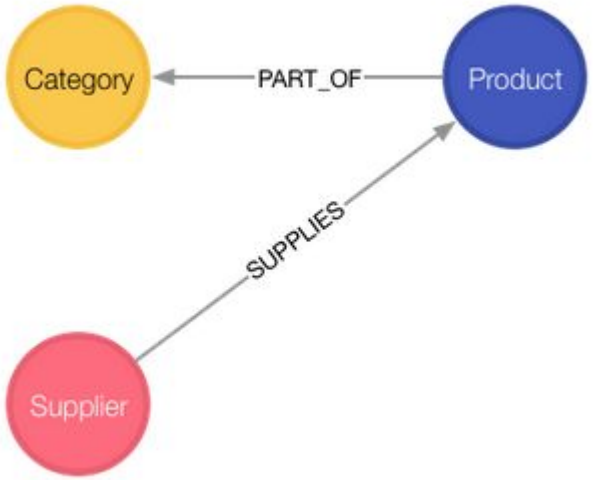
```
LOAD CSV WITH HEADERS FROM
"file:///suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Trabalhando com o banco Northwind

Crie índices para os campos ID de cada tipo de vértices

```
CREATE INDEX ON :Product(productID)
CREATE INDEX ON :Category(categoryID)
CREATE INDEX ON :Supplier(supplierID)
```

Trabalhando com o banco Northwind



Os products, categories e suppliers estão relacionados através de uma chave estrangeira no banco relacional. Vamos criar estes relacionamentos no grafo.

Trabalhando com o banco Northwind

Criando relacionamentos entre produtos, categoria do tipo

"PART_OF" e produtos, suppliers do tipo "SUPPLIES"

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

Observe que este tipo de comparação realizada ("join") é feita apenas para criação do relacionamento.

Desafio 1: Realizando consultas

Trabalhando com o banco Northwind

Realizando algumas consultas: Listar as categorias de produtos fornecidas por cada fornecedor.

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
```

Trabalhando com o banco Northwind

Realizando algumas consultas: Listar as categorias de produtos fornecidas por cada fornecedor.

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
```

O símbolo (-->) significa “related to”, ou seja, não importa o tipo do relacionamento. Sendo o símbolo (--), não importaria também a direção.

Trabalhando com o banco Northwind

Realizando algumas consultas: **Listar as categorias** de produtos fornecidas por cada fornecedor.

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
```


Retorna uma lista contendo os valores retornados por uma expressão. O uso dessa função agrega dados em uma única lista.

Trabalhando com o banco Northwind

Realizando algumas consultas: Encontrar o fornecedor de cada produto.

```
MATCH (c:Category {categoryName:"Produce"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.companyName as ProduceSuppliers
```

Para saber mais

Para saber mais sobre importação de dados no Neo4j, veja no site dos desenvolvedores (link abaixo)

Tutorial: Import Data Into Neo4j (importação do Northwind completo)
https://neo4j.com/developer/guide-importing-data-and-etl/

Dica com importação de arquivos

No Neo4j é possível importar arquivos diretamente da internet, veja:

```
LOAD CSV WITH HEADERS FROM
"http://data.neo4j.com/northwind/categories.csv"
AS row
CREATE (n:Category)
SET n = row
```

Importando de um arquivo .csv diretamente da internet

Para saber mais

Acesse o site oficial do Neo4j

https://neo4j.com/developer/guide-importing-data-and-etl/

Desafio 2: Movie Dataset

Criando o banco de dados

Crie um banco de dados de filmes. Utilize o script chamado “movies.txt” fornecido no módulo da disciplina, importe e execute no Neo4j

Veja o grafo de resultado

Executando algumas consultas

Encontre filmes lançados nos anos 90...

```
MATCH (nineties:Movie) WHERE nineties.released >= 1990 AND
nineties.released < 2000 RETURN nineties.title
```

Quais pessoas estão relacionadas ao "Cloud Atlas"...

```
MATCH (people:Person)-[relatedTo]-(:Movie
{title: "Cloud Atlas"}) RETURN people.name,
Type(relatedTo), relatedTo
```

Shortest Path Query

Banco de dados de Grafos nos permite realizar, de forma simples, consultas do tipo “caminho mais curto” entre dois vértices.

Vamos criar um dataset de exemplo de filme e seus atores/diretores para ilustrar nosso exemplo.

Shortest Path Query

Mostrar films e atores até 3 “saltos" de Kevin Bacon

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..3]-(hollywood)
RETURN DISTINCT hollywood
```

Mostrar o caminho mais curto de Kevin Bacon até Meg Ryan (independente do relacionamento)

```
MATCH p=shortestPath(
(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Shortest Path Query

Observe no grafo exibido como resultado a distância, em relação ao número de vértices, de Kevin Bacon para Meg Ryan

Fazendo recomendações

Uma das funcionalidades de um banco de dados de grafos é permitir realizar recomendações

Veja nos exemplos a seguir:

Fazendo recomendações

Mostrar os co-atores do Tom Hanks (atores que trabalharam com ele) para encontrar os co-co-atores que jamais trabalharam com Tom Hanks...

ideia: encontrar atores que nunca trabalharam com Tom Hanks, mas seus co-atores já trabalharam (quem trabalhou com quem trabalhou com Tom Hanks)

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->()<-[:ACTED_IN]-(cocoActors) AND tom <> cocoActors
RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC
```

Fazendo recomendações

Observe na resposta que “Tom Cruise” trabalhou em 5 filmes com atores que trabalharam com Tom Hanks, porém os dois jamais trabalharam juntos, até então.

Fazendo recomendações

Encontrar alguém que possa apresentar Tom Hanks para Tom Cruise

ideia: encontrar alguém que possa apresentar Tom Hacks com um potencial co-ator.

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

Fazendo recomendações

Observe no grafo exibido como resultado da consulta quem são os atores que poderia apresentar Tom Cruise a Tom Hanks.

A ideia é que se, por exemplo, Tom Cruise trabalhou com vários atores que trabalharam com Tom Hanks, então Tom Cruise poderia ser um potencial “companheiro de cena” para Tom Hanks.

Neo4j e Python

Neo4j e Python

A biblioteca GraphDatabase do Python é utilizado para conectar um programa em Python com o Neo4j (banco de dados de grafos).

```
> pip install neo4j-driver
```

Comando para a instalação da biblioteca no Python.

```
> pip install neo4jrestclient
```

Referências: https://neo4j.com/developer/python/

http://neo4j-rest-client.readthedocs.io/en/latest/info.html

Neo4j e Python

Crie usuário e relacionamento entre eles, conforme o código abaixo:

String de conexão com o banco

```
from neo4jrestclient.client import GraphDatabase
db = GraphDatabase("http://localhost:7474", username="neo4j", password="P@ssw0rd")

user = db.labels.create("Usuario")

u1 = db.nodes.create(name="Bob")
u2 = db.nodes.create(name="Alice")
u3 = db.nodes.create(name="Lea")
u4 = db.nodes.create(name="Ana")
u5 = db.nodes.create(name="Joel")
```

Criação do label “Usuario”

Criação dos nós do label “Usuário”

Neo4j e Python

Crie usuário e relacionamento entre eles, conforme o código abaixo:

Adiciona os nós criados

```
user.add(u1, u2, u3, u4, u5)
```

Criação dos relacionamentos (arestas) entre os nós.

```
u1.relationships.create("follows", u2)
u4.relationships.create("follows", u1)
u2.relationships.create("follows", u3)
u2.relationships.create("follows", u5)
```

As arestas representam os relacionamentos entre os usuários como ocorre em um rede social.

Neo4j e Python

Programa em Python que retorna todos os usuários que Bob segue na rede.

```
from neo4jrestclient.client import GraphDatabase
from neo4jrestclient import client

db = GraphDatabase("http://localhost:7474", username="neo4j", password="P@ssw0rd")

# Obtendo todos os usuário que Bob segue na rede
q = 'MATCH (u1:Usuario)-[r:follows]->(u2:Usuario) WHERE u1.name="Bob" RETURN u1,
type(r), u2'

results = db.query(q, returns=(client.Node, str, client.Node))
```

Neo4j e Python

Programa em Python que retorna todos os usuários que Bob segue na rede.

```
for r in results:
     print("(%s)-[%s]->(%s)" % (r[0]["name"], r[1], r[2]["name"]))

# saída:
# (Bob)-[follows]->(Alice)
```

Quando usar e não usar

Situações apropriadas para o uso

Dados conectados: Redes sociais são exemplos clássicos do uso de Grafos. Além de representar amigos, podem também representar relacionamento entre funcionários, clientes, etc.

Referências: Pramod J.; Sadalage, Martin Fowler. NoSQL Essencial. 2013

Situações apropriadas para o uso

Roteamento, serviços de localização: dois pontos (locais) podem ser conectados a uma determinada distância. Tal informação pode ser útil para recomendar um restaurante mais próximo. Ou qualquer serviço baseado em localização.

Referências: Pramod J.; Sadalage, Martin Fowler. NoSQL Essencial. 2013

Situações apropriadas para o uso

Sistema de recomendação: Pode ser feita recomendações do tipo “seus amigos também compraram este produto” ou “quais itens são sempre comprados juntos?”

Localização: “quem visita a cidade A também visitou a B”

Referências: Pramod J.; Sadalage, Martin Fowler. NoSQL Essencial. 2013

Situações para *não* usar

Geralmente quando há muita atualização das entidades. Alterar propriedade em todos os nós pode ser lento dependendo do tamanho do grafo.

Problemas que lidam com o grafo inteiro podem ser muito lentos.

Referências: Pramod J.; Sadalage, Martin Fowler. NoSQL Essencial. 2013

Links para saber mais:

Para conhecer mais sobre tipos de bancos NoSQL:

http://nosql-database.org/

Dicas!

Para saber mais: Consulte o site dos desenvolvedores

https://neo4j.com

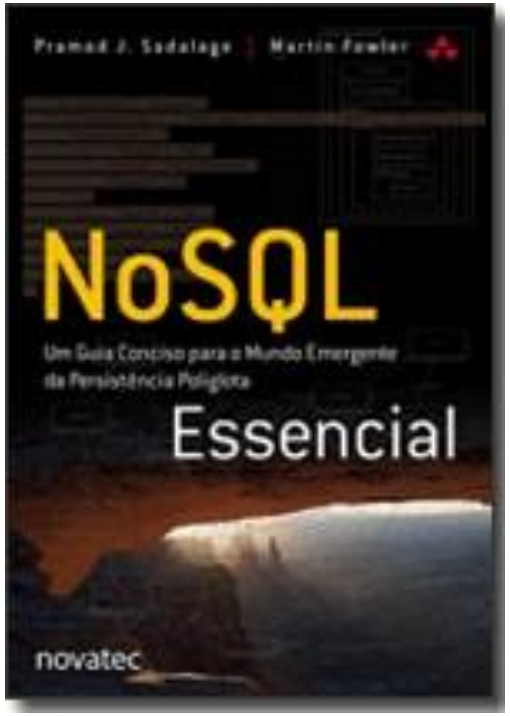
Para saber mais sobre drive do Neo4j para Python:

https://neo4j.com/developer/python/

Para saber mais: visite o canal oficial do Neo4j no YouTube:

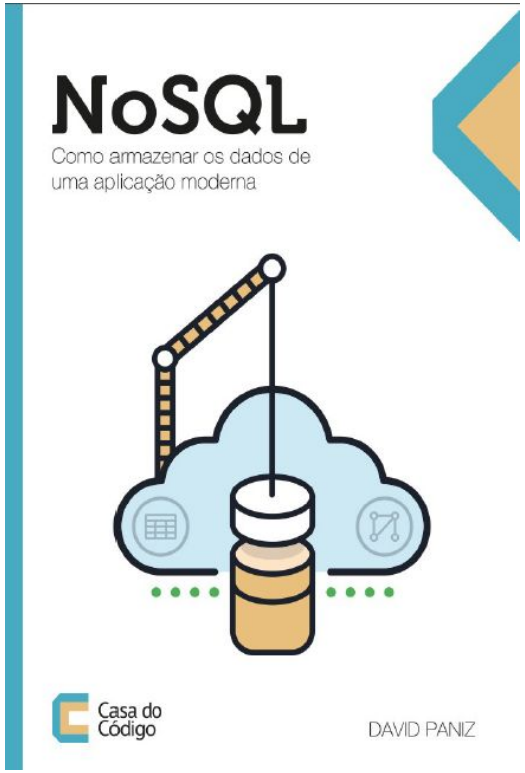
https://www.youtube.com/Neo4j

Principais Referências



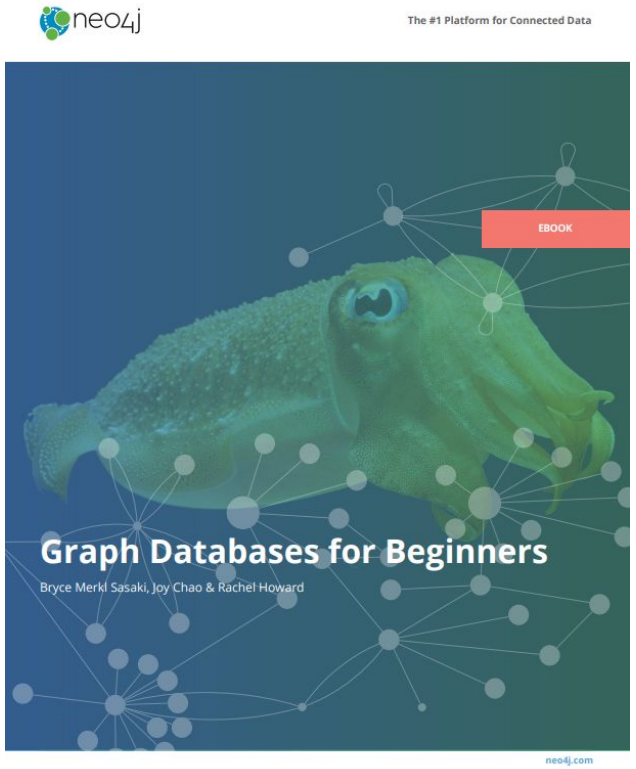
Pramod J.; Sadalage, Martin Fowler. **NoSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. Novatec Editora, 2013.

Principais Referências



Paniz, David. NoSQL: **Como armazenar os dados de uma aplicação moderna**. Casa do Código, 2017.

Principais Referências



Sasaki, B.; Chao, J.; Howard, R..
Graph Databases for Beginners.
Neo4j.com.

Download gratuito em:
https://neo4j.com/whitepapers/graph-databases-beginners-ebook/?r