# 7 A ascensão dos bancos de dados gráficos

## Este capítulo cobre

- ÿ Apresentando dados conectados e como eles estão relacionados para gráficos e bancos de dados gráficos ÿ
- Aprendendo como os bancos de dados gráficos diferem dos bancos de dados relacionais
- ÿ Descobrindo o banco de dados gráfico Neo4j ÿ
- Aplicando o processo de ciência de dados a um projeto de mecanismo de recomendação com o banco de dados gráfico Neo4j

Enquanto, por um lado, produzimos dados em grande escala, levando empresas como Google, Amazon e Facebook a encontrarem formas inteligentes de lidar com esta questão, por outro lado, deparamo-nos com dados que se estão a tornar mais interligados do que nunca. . Gráficos e redes estão presentes em nossas vidas. Ao apresentar vários exemplos motivadores, esperamos ensinar ao leitor como reconhecer um problema gráfico quando ele se revela. Neste capítulo, veremos como aproveitar essas conexões ao máximo usando um banco de dados gráfico e demonstraremos como usar o Neo4j, um banco de dados gráfico popular.

## 7.1 Apresentando dados conectados e bancos de dados gráficos

Vamos começar nos familiarizando com o conceito de dados conectados e sua representação como dados gráficos.

ÿ *Dados conectados* — *Como* o nome indica, os dados conectados são caracterizados pelo fato de que os dados em questão possuem um relacionamento que os torna
conectados. ÿ Gráficos — Frequentemente referidos na mesma frase como dados conectados. Os gráficos são adequados para representar a conectividade dos dados de uma
forma significativa. ÿ Bancos de dados *gráficos* – introduzidos no capítulo 6. A razão pela qual este assunto merece atenção especial é porque, além do fato de os dados estarem aumentando em tamanho, eles também estão se tornando mais interconectados. Não é necessário muito esforço para encontrar exemplos bem conhecidos de dados conectados.

Um exemplo proeminente de dados que assumem a forma de rede são os dados de mídia social. As mídias sociais nos permitem compartilhar e trocar dados em redes, gerando assim uma grande quantidade de dados conectados. Podemos ilustrar isso com um exemplo simples. Vamos supor que temos duas pessoas em nossos dados, Usuário1 e Usuário2. Além disso, sabemos o nome e o sobrenome do Usuário1 (nome: Paul e sobrenome: Beun) e do Usuário2 (nome: Jelme e sobrenome: Ragnar). Uma forma natural de representar isto poderia ser desenhando-o num quadro branco, como mostra a figura 7.1.

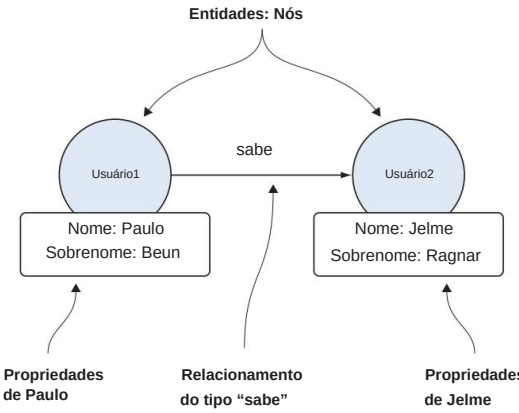


**Figura 7.1 Um exemplo simples de dados conectados: duas entidades ou nós (Usuário1, Usuário2), cada um com propriedades (nome, sobrenome), conectados por um relacionamento (sabe)**

A terminologia da figura 7.1 é descrita abaixo:

ÿ *Entidades* — Temos duas entidades que representam pessoas (Usuário1 e Usuário2). Estas entidades possuem as propriedades “nome” e “sobrenome”. ÿ
*Propriedades* — *As* propriedades são definidas por pares de valores-chave. A partir deste gráfico também podemos inferir que o Usuário1 com a propriedade *“nome”* Paul conhece o Usuário2 com a propriedade *“nome”* Jelme.

ÿ *Relacionamentos—Este* é o relacionamento entre Paul e Jelme. Note que a relação tem um rumo: é Paul quem *"conhece"* Jelme e não o contrário. Usuário1 e Usuário2 representam pessoas e, portanto, podem ser agrupados. ÿ *Rótulos — Em* um banco de dados gráfico, é possível agrupar nós usando rótulos. Usuário1 e Usuário2 poderiam, neste caso, ser rotulados como "Usuário".

Os dados conectados geralmente contêm muito mais entidades e conexões. Na figura 7.2 podemos ver um gráfico mais extenso. Estão incluídas mais duas entidades: Country1 com o nome Camboja e Country2 com o nome Suécia. Existem mais dois relacionamentos: "Has_been_in" e "Is_born_in". No gráfico anterior apenas as entidades incluíam uma propriedade, agora os relacionamentos também contêm uma propriedade. Esses gráficos são conhecidos como gráficos de propriedades. O relacionamento que conecta os nós Usuário1 e País1 é do tipo "Has_been_in" e tem como propriedade "Data" que representa um valor de dado. Da mesma forma, o Usuário2 está conectado ao País2, mas por meio de um tipo de relacionamento diferente, que é do tipo "Nasce_em". Observe que os tipos de relacionamentos nos fornecem um contexto dos relacionamentos entre os nós. Os nós podem ter vários relacionamentos.

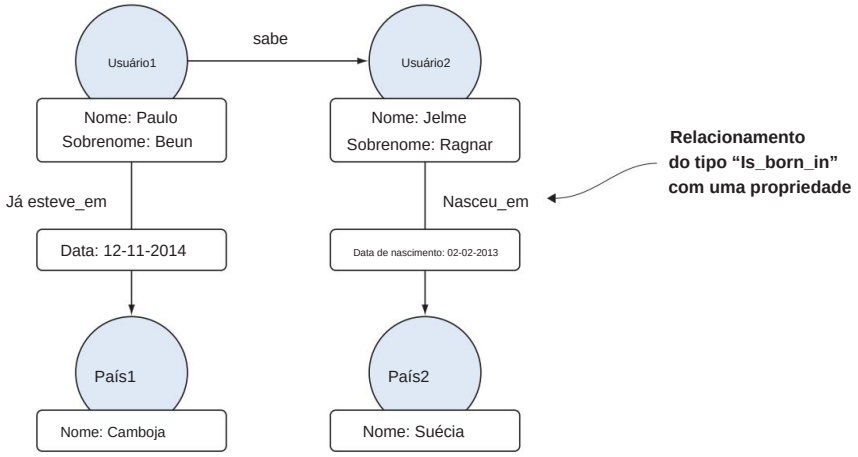


**Figura 7.2 Um exemplo de dados conectados mais complicado onde mais duas entidades foram incluídas (País1 e País2) e dois novos relacionamentos ("Has_been_in" e "Is_born_in")**

Esse tipo de representação de nossos dados nos oferece uma maneira intuitiva de armazenar dados conectados. Para explorar nossos dados, precisamos percorrer o gráfico seguindo caminhos predefinidos para encontrar os padrões que procuramos. E se alguém quiser saber onde Paulo esteve? Traduzido para a terminologia de banco de dados gráfico, gostaríamos de encontrar o padrão "Paul esteve dentro". Para responder a isso, começaríamos no nó com o

nomeie “Paul” e atravesse para o Camboja através do relacionamento “Has_been_in”. Daí um a travessia do gráfico, que corresponde a uma consulta ao banco de dados, seria a seguinte:

**1** *Um nó inicial - neste* caso, o nó com propriedade de nome “Paul”

**2** *Um caminho de travessia - Neste* caso, um caminho começando no nó Paul e indo para o Camboja

**3** Nó *final* —Nó de país com propriedade de nome “Camboja”

Para entender melhor como os bancos de dados gráficos lidam com dados conectados, é apropriado para expandir um pouco mais os gráficos em geral. Os gráficos são extensivamente estudados no domínios da ciência da computação e matemática em um campo chamado teoria dos grafos. Gráfico teoria é o estudo de gráficos, onde os gráficos representam as estruturas matemáticas usado para modelar relações de pares entre objetos, conforme mostrado na figura 7.3. O que faz o O que os torna tão atraentes é que eles têm uma estrutura que se presta à visualização de dados conectados. Um gráfico é definido por vértices (também conhecidos como nós no banco de dados de gráficos mundo) e bordas (também conhecidas como relacionamentos). Esses conceitos formam os fundamentos básicos nos quais as estruturas de dados gráficos são baseadas.

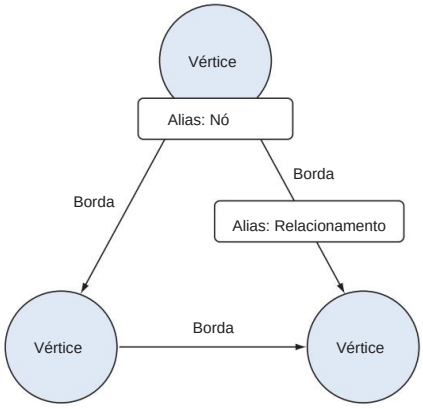


**Figura 7.3 Em sua essência, um grafo consiste em nós (também conhecidos como vértices) e arestas (que conectam os vértices), conforme conhecido pela definição matemática de um grafo. Essas coleções de objetos representam o gráfico.**

Em comparação com outras estruturas de dados, uma característica distintiva dos dados conectados é a sua natureza não linear: qualquer entidade pode ser conectada a qualquer outra através de uma variedade de relacionamentos. tipos e entidades e caminhos intermediários. Nos gráficos, você pode fazer uma subdivisão entre gráficos direcionados e não direcionados. As arestas de um gráfico direcionado têm - como poderia ser de outra forma - uma direção. Embora se possa argumentar que todo problema poderia de alguma forma ser representado como um problema gráfico, é importante entender quando é ideal fazê-lo e quando não é.

### 7.1.1 Por que e quando devo usar um banco de dados gráfico?

A busca de determinar qual banco de dados de gráficos deve ser usado pode ser uma tarefa complicada. processo a ser realizado. Um aspecto importante neste processo de tomada de decisão é

encontrar a representação correta para seus dados. Desde o início da década de 1970, o tipo mais comum de banco de dados em que se podia confiar era o relacional. Mais tarde, surgiram outros, como o banco de dados hierárquico (por exemplo, IMS) e o parente mais próximo do banco de dados gráfico: o banco de dados de rede (por exemplo, IDMS). Mas durante as últimas décadas o cenário tornou-se muito mais diversificado, dando aos usuários finais mais opções dependendo de suas necessidades específicas. Considerando o desenvolvimento recente dos dados que está se tornando disponível, duas características merecem ser destacadas aqui. O primeiro é o tamanho dos dados e o outro a complexidade dos dados, pois mostrado na figura 7.4.

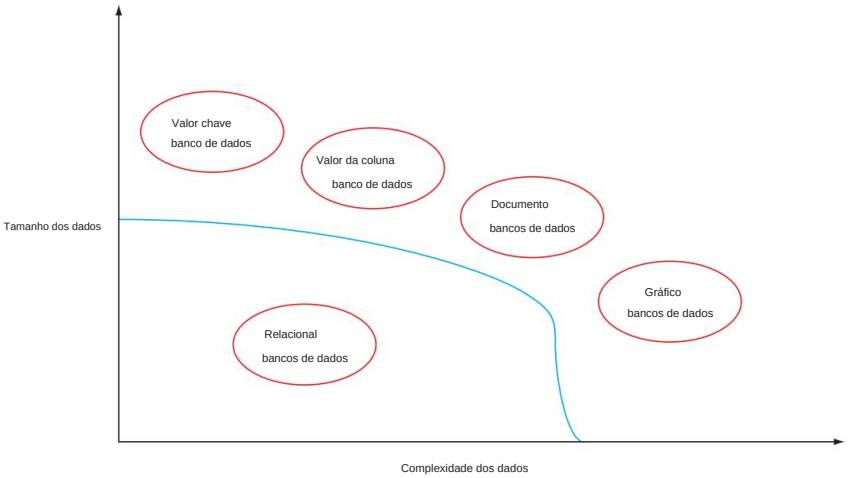


**Figura 7.4 Esta figura ilustra o posicionamento de bancos de dados gráficos em um espaço bidimensional onde uma dimensão representa o tamanho dos dados com os quais se está lidando e a outra dimensão representa a complexidade em termos de quão conectados os dados estão. Quando os bancos de dados relacionais não conseguem mais lidar com a complexidade de um conjunto de dados devido à sua conectividade, mas não ao seu tamanho, os bancos de dados gráficos podem ser sua melhor opção.**

Como indica a figura 7.4, precisaremos contar com um banco de dados gráfico quando os dados forem complexos, mas ainda pequenos. Embora “pequeno” seja algo relativo aqui, ainda estamos falando de centenas de milhões de nós. Lidar com a complexidade é o principal ativo de um banco de dados gráfico e o “porquê” final para você usá-lo. Para explicar que tipo de *complexidade* se entende aqui, primeiro pense em como funciona um banco de dados relacional tradicional.

Ao contrário do que o nome dos bancos de dados relacionais indica, pouco há de relacional neles, exceto que as chaves estrangeiras e as chaves primárias são o que se relacionam. tabelas. Em contraste, os relacionamentos em bancos de dados gráficos são cidadãos de primeira classe. Através Nesse aspecto, eles se prestam bem à modelagem e consulta de dados conectados. A

o banco de dados relacional preferiria se esforçar para minimizar a redundância de dados. Este processo é conhecido como normalização de banco de dados, onde uma tabela é decomposta em tabelas (menos redundantes), mantendo todas as informações intactas. Em um banco de dados normalizado é necessário realizar alterações de um atributo em apenas uma tabela. O O objetivo deste processo é isolar alterações de dados em uma tabela. Os sistemas de gerenciamento de banco de dados relacional (RDBMS) são uma boa escolha como banco de dados para dados que se ajustam perfeitamente em um formato tabular. As relações nos dados podem ser expressas juntando-se os tabelas. Seu ajuste começa a diminuir quando as junções se tornam mais complicadas, especialmente quando se tornam junções muitos-para-muitos. O tempo de consulta também aumentará quando o tamanho dos seus dados começa a aumentar e manter o banco de dados será um desafio maior. Esses fatores prejudicarão o desempenho do seu banco de dados. Os bancos de dados gráficos, por outro lado, armazenam dados inerentemente como nós e relacionamentos. Embora bancos de dados gráficos são classificados como um tipo de banco de dados NoSQL , uma tendência para apresentá-los como uma categoria por direito próprio existe. Procura-se a justificativa para isso observando que os outros tipos de bancos de dados NoSQL são orientados para agregação, enquanto os bancos de dados gráficos não são.

Um banco de dados relacional pode, por exemplo, ter uma tabela representando “pessoas” e suas propriedades. Qualquer pessoa está relacionada com outras pessoas através de parentesco (e amizade, e assim por diante); cada linha pode representar uma pessoa, mas conectando-as a outras linhas na tabela de pessoas seria uma tarefa imensamente difícil. Você adiciona uma variável que contém o identificador exclusivo do primeiro filho e um extra para conter o ID do o segundo filho? Onde você para? Décimo filho?

Uma alternativa seria usar uma tabela intermediária para relacionamentos entre pais e filhos, mas você precisará de uma tabela separada para outros tipos de relacionamento, como amizade. Em neste último caso você não obtém proliferação de colunas, mas sim proliferação de tabelas: uma tabela de relacionamento para cada tipo de relacionamento. Mesmo que você de alguma forma consiga modelar o dados de forma que todas as relações familiares estejam presentes, você precisará de consultas difíceis para obter a resposta a perguntas simples como “Gostaria que os netos de John McBain.” Primeiro você precisa encontrar os filhos de John McBain. Depois de encontrar seus filhos, você precisa encontrar o deles. Quando você encontrar todos os netos, você terá atingido o tabela de “pessoas” três vezes:

**1** Encontre McBain e busque seus filhos.

**2** Procure as crianças com as identificações que você obteve e obtenha as identificações dos filhos delas.

**3** Encontre os netos de McBain.

A Figura 7.5 mostra as pesquisas recursivas em um banco de dados relacional necessárias para obter John McBain para seus netos se tudo estiver em uma única mesa.

A Figura 7.6 é outra maneira de modelar os dados: o relacionamento pai-filho é uma tabela separada.

Pesquisas recursivas como essas são ineficientes, para dizer o mínimo.

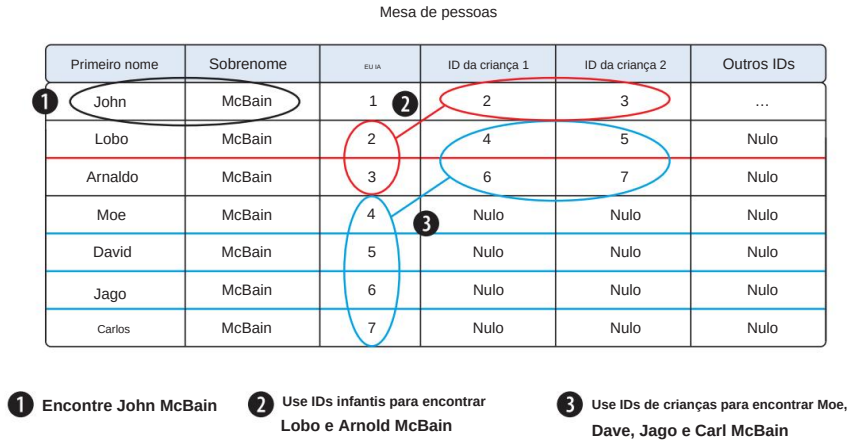


**Figura 7.5 Pesquisa recursiva versão 1: todos os dados em uma tabela**

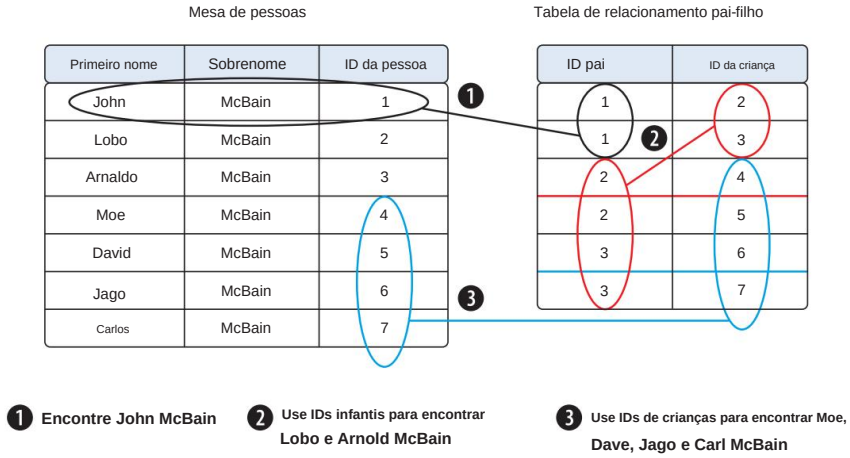


**Figura 7.6 Pesquisa recursiva versão 2: usando uma tabela de relacionamento pai-filho**

Os bancos de dados gráficos brilham quando esse tipo de *complexidade* surge. Vejamos os mais populares entre eles.

## 7.2 Apresentando Neo4j: um banco de dados gráfico

Os dados conectados geralmente são armazenados em bancos de dados gráficos. Esses bancos de dados são especificamente projetado para lidar com a estrutura de dados conectados. A paisagem disponível bancos de dados gráficos são bastante diversos atualmente. Os três mais conhecidos em ordem de

popularidade decrescente são Neo4j, OrientDb e Titan. Para mostrar nosso estudo de caso escolheremos o mais popular no momento da escrita (veja http://db-engines .com/en/ranking/graph+dbms, setembro de 2015).

Neo4j é um banco de dados gráfico que armazena os dados em um gráfico contendo nós e relacionamentos (ambos podem conter propriedades). Este tipo de banco de dados gráfico é conhecido como gráfico de propriedades e é adequado para armazenar dados conectados. Possui um esquema flexível que nos dará liberdade para alterar nossa estrutura de dados se necessário, fornecendo nos a capacidade de adicionar novos dados e novos relacionamentos, se necessário. É um código aberto projeto, tecnologia madura, fácil de instalar, fácil de usar e bem documentado. Neo4j também possui uma interface baseada em navegador que facilita a criação de gráficos para fins de visualização. Para acompanhar, este seria o momento certo para instalar o Neo4j. Neo4j pode ser baixado em http://neo4j.com/download/. Todas as etapas necessárias para uma instalação bem-sucedida estão resumidas no apêndice C.

Agora vamos apresentar as quatro estruturas básicas do Neo4j:

- ÿ Nós — representam entidades como documentos, usuários, receitas e assim por diante. Certo propriedades podem ser atribuídas aos nós.
- ÿ *Relacionamentos — existem* entre os diferentes nós. Eles podem ser acessados ou autônomo ou por meio dos nós aos quais estão anexados. Os relacionamentos também podem contêm propriedades, daí o nome modelo de gráfico de propriedades. Todo relacionamento tem um nome e uma direção, que juntos fornecem contexto semântico para o nós conectados pelo relacionamento.
- ÿ *Propriedades — tanto* os nós quanto os relacionamentos podem ter propriedades. As propriedades são definido por pares de valores-chave.
- ÿ *Rótulos* —Podem ser usados para agrupar nós semelhantes para facilitar uma passagem mais rápida através gráficos.

Antes de realizar uma análise, um bom hábito é projetar cuidadosamente seu banco de dados para que ele se ajusta às consultas que você gostaria de realizar ao realizar sua análise. Gráfico os bancos de dados têm a característica agradável de serem compatíveis com o quadro branco. Se alguém tentar para desenhar a configuração do problema em um quadro branco, este desenho será muito parecido com o projeto de banco de dados para o problema definido. Portanto, tal desenho de quadro branco seria então um bom ponto de partida para projetar nosso banco de dados.

Agora, como recuperar os dados? Para explorar nossos dados, precisamos percorrer o gráfico seguindo caminhos predefinidos para encontrar os padrões que procuramos. O O navegador Neo4j é um ambiente ideal para criar e brincar com seus usuários conectados. dados até chegar ao tipo certo de representação para consultas ideais, conforme mostrado em figura 7.7. O esquema flexível do banco de dados gráfico nos serve bem aqui. Nisso navegador, você pode recuperar seus dados em linhas ou como um gráfico. O Neo4j possui sua própria linguagem de consulta para facilitar a criação e recursos de consulta de gráficos.

Cypher é uma linguagem altamente expressiva que compartilha o suficiente com SQL para aprimorar o processo de aprendizagem da língua. Na seção a seguir, criaremos nosso próprio dados usando Cypher e insira-os no Neo4j. Então podemos brincar com os dados.

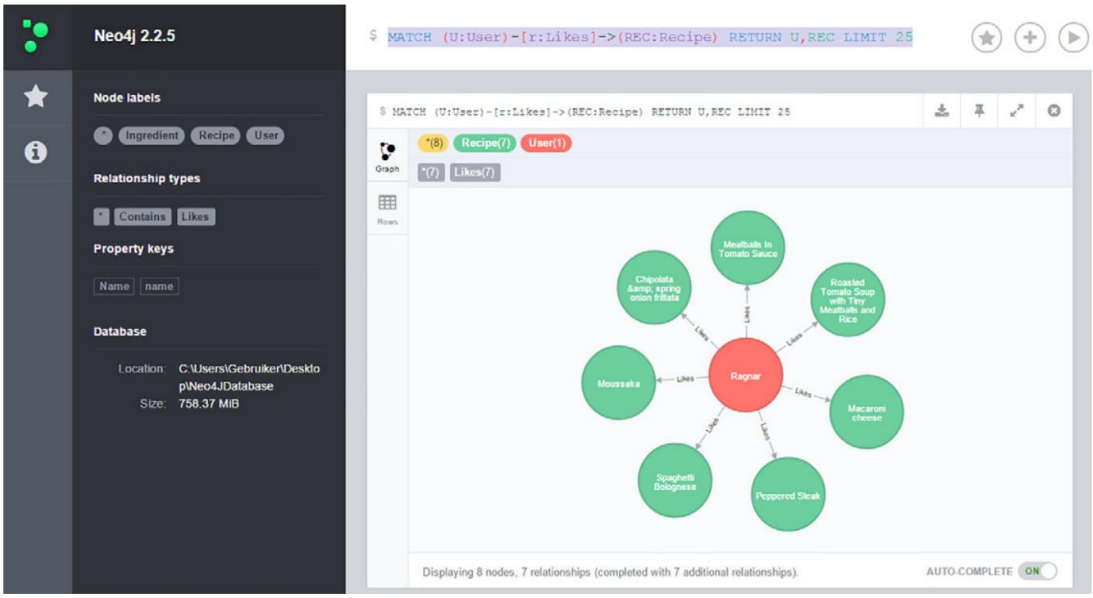


**Figura 7.7 Interface do Neo4j 2.2.5 com consulta resolvida do capítulo de estudo de caso**

### 7.2.1 Cypher: uma linguagem de consulta gráfica

Vamos apresentar o Cypher e sua sintaxe básica para operações gráficas. A ideia desta seção é apresentar o suficiente sobre o Cypher para começarmos a usar o navegador Neo4j. No no final desta seção, você poderá criar seus próprios dados conectados usando Cypher no navegador Neo4j e execute consultas básicas para recuperar os resultados da consulta. Para uma introdução mais extensa ao Cypher, você pode visitar http://neo4j.com/docs/estável/cypher-query-lang.html. Começaremos desenhando um gráfico social simples acompanhado de uma consulta básica para recuperar um padrão predefinido como exemplo. Na próxima etapa desenharemos um gráfico mais complexo que nos permitirá usar consultas mais complicadas em Cifra. Isso nos ajudará a nos familiarizar com Cypher e nos guiará no caminho para trazendo nosso caso de uso para a realidade. Além disso, mostraremos como criar nossos próprios dados conectados simulados usando Cypher.

A Figura 7.8 mostra um grafo social simples de dois nós, conectados por uma relação de digite “sabe”. Os nós possuem as propriedades “nome” e “sobrenome”.

Agora, se quisermos descobrir o seguinte padrão: “Quem Paulo conhece?” qua consulte isso usando Cypher. Para encontrar um padrão no Cypher, começaremos com uma cláusula Match . Em

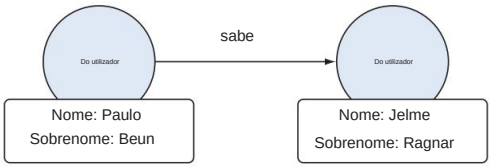


**Figura 7.8 Um exemplo de gráfico social simples com dois usuários e um relacionamento**

Nesta consulta iniciaremos a busca no nó Usuário com a propriedade de nome “Paul”. Observe como o nó está entre parênteses, conforme mostrado no trecho de código abaixo, e o relacionamento está entre colchetes. Os relacionamentos são nomeados com um prefixo de dois pontos (:) e a direção é descrita por meio de setas. O espaço reservado p2 conterá todos os nós de usuário que possuem o relacionamento do tipo “conhece” como um relacionamento de entrada.
Com a cláusula return podemos recuperar os resultados da consulta.

```
Match(p1:Usuário { nome: 'Paul' } )-[:sabe]->(p2:Usuário)
Retornar p2.nome
```

Observe a estreita relação entre como formulamos nossa pergunta verbalmente e a maneira como o banco de dados gráfico traduz isso em uma travessia. No Neo4j, essa impressionante expressividade é possível graças à sua linguagem de consulta gráfica, Cypher.

Para tornar os exemplos mais interessantes, vamos supor que nossos dados sejam representados pelo gráfico da figura 7.9.

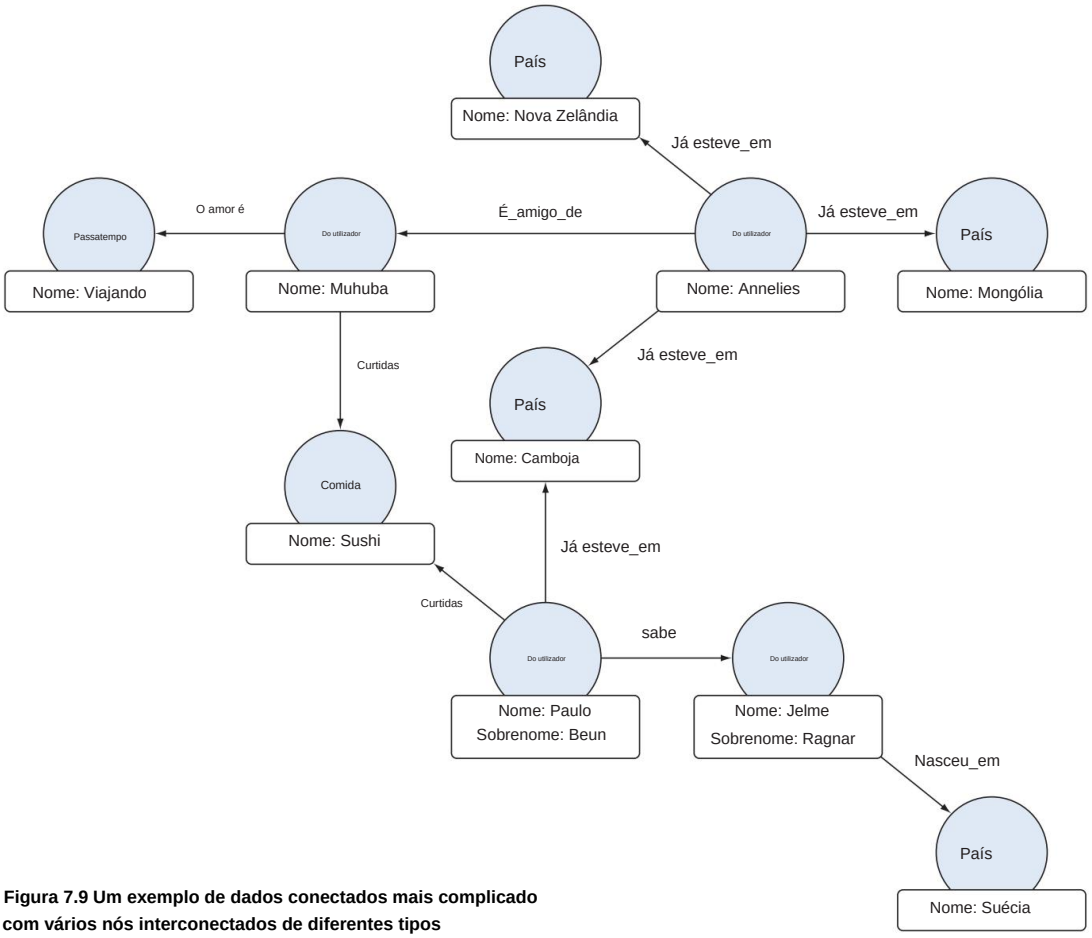


**Figura 7.9 Um exemplo de dados conectados mais complicado com vários nós interconectados de diferentes tipos**

Podemos inserir os dados conectados na figura 7.9 no Neo4j usando Cypher. Pudermos escreva comandos Cypher diretamente na interface baseada em navegador do Neo4j ou, alternativamente, por meio de um driver Python (consulte http://neo4j.com/developer/python/ para uma visão geral). Esta é uma boa maneira de ter uma experiência prática com dados e gráficos conectados bancos de dados.

Para escrever uma instrução create apropriada em Cypher, primeiro devemos ter um bom compreensão de quais dados gostaríamos de armazenar como nós e quais como relacionamentos, quais deveriam ser suas propriedades e se os rótulos seriam úteis. A primeira decisão é decidir quais dados devem ser considerados como nós e quais como relacionamentos para forneça um contexto semântico para esses nós. Na figura 7.9 escolhemos representar os usuários e países em que estiveram como nós. Dados que fornecem informações sobre um nó específico, por exemplo, um nome associado a um nó, pode ser representado como uma propriedade. Todos os dados que fornecem contexto sobre dois ou mais nós serão considerado como um relacionamento. Os nós que partilham características comuns, por exemplo Camboja e Suécia são ambos países, também serão agrupados através de rótulos. Na figura 7.9 isso já está feito.

Na listagem a seguir demonstramos como os diferentes objetos podem ser codificados no Cypher por meio de uma grande instrução create. *Esteja ciente de que Cypher diferencia maiúsculas de minúsculas.*

**Listagem 7.1 Instrução de criação de dados Cypher**

```
CRIAR (usuário1:Usuário {nome:'Annelies'}),
  (usuário2:Usuário {nome:'Paul' , Sobrenome: 'Beun'}),
  (usuário3:Usuário {nome:'Muhuba'}),
  (usuário4:Usuário {nome: 'Jelme' , Sobrenome: 'Ragnar'}),
  (país1:País {nome:'Mongólia'}),
  (país2:País {nome:'Camboja'}),
  (país3:País {nome:'Nova Zelândia'}),
  (país4:País {nome:'Suécia'}),
  (comida1:Comida {nome:'Sushi' }),
  (hobby1:Hobby {nome:'Viagem'}),
  (usuário1)-[:Já esteve_em]->(país1),
  (usuário1)-[: Já esteve_em]->(país2),
  (usuário1)-[: Já esteve_em]->(país3),
  (usuário2)-[: Já esteve_em]->(país2),
  (usuário1)-[:É_mãe_de]->(usuário4),
  (usuário2)-[: sabe]->(usuário4),
  (usuário1)-[: É_amigo_de]->(usuário3),
  (usuário2)-[: Curtidas]->(comida1),
  (usuário3)-[: Curtidas]->(comida1),
  (usuário4)-[: Nasceu_em]->(país4)
```

Executar esta instrução create de uma só vez tem a vantagem de que o sucesso desta execução nos garantirá que o banco de dados gráfico foi criado com sucesso. Se um erro existir, o gráfico não será criado.

Num cenário real, devem-se também definir índices e restrições para garantir um rápido pesquisar e não pesquisar em todo o banco de dados. Não fizemos isso aqui porque nosso o conjunto de dados simulados é pequeno. No entanto, isso pode ser feito facilmente usando Cypher. Consulte o

Documentação Cypher para saber mais sobre índices e restrições (http://neo4j.com/docs/stable/cypherdoc-labels-constraints-and-indexes.html). Agora que já criamos nossos dados, podemos consultá-los. A consulta a seguir retornará todos os nós e relacionamentos no banco de dados:

```
CORRESPONDÊNCIA (n)-[r]-()
RETORNAR n,r
```

**Encontre todos os nós (n) e todos os seus relacionamentos [r].**

**Mostre todos os nós n e todos os relacionamentos r.**

A Figura 7.10 mostra o banco de dados que criamos. Podemos comparar este gráfico com o gráfico que imaginamos em nosso quadro branco. Em nosso quadro branco agrupamos nós de pessoas em um rótulo "Usuário" e nós de países em um rótulo "País". Apesar de nós nesta figura não são representados por seus rótulos, os rótulos estão presentes em nosso base de dados. Além disso, também sentimos falta de um nó (Hobby) e de um relacionamento do tipo "O amor é". Eles podem ser facilmente adicionados através de uma instrução merge que criará o nó e relacionamento se ainda não existirem:

```
Mesclar (usuário3)-[:Ama]->(hobby1)
```

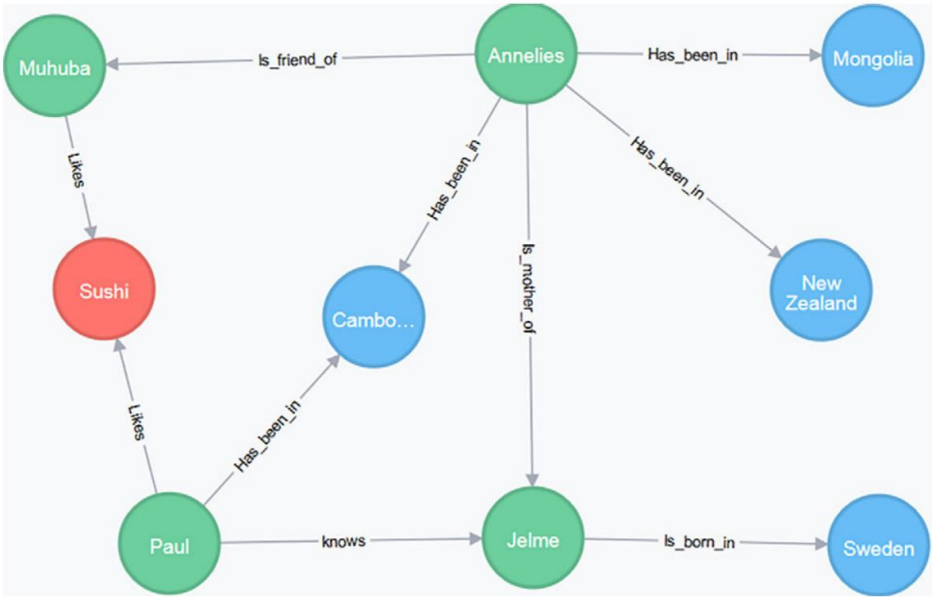


**Figura 7.10 O gráfico desenhado na figura 7.9 agora foi criado na interface web do Neo4j. Os nós não são representados pelos seus rótulos, mas pelos seus nomes. Podemos inferir pelo gráfico que falta o rótulo Hobby com o nome Viajar. A razão para isso é porque esquecemos de incluir este nó e seu relacionamento correspondente na instrução create.**

Podemos fazer muitas perguntas aqui. Por exemplo:

- ÿ Pergunta 1: Que países Annelies visitou? O código Cypher para criar a resposta (mostrado na figura 7.11) é

```
Match(u:Usuário{nome:'Annelies'}) – [:Has_been_in]-> (c:Country)
Retorne u.nome, c.nome
```

- ÿ Pergunta 2: Quem esteve onde? O código Cypher (explicado na figura 7.12) é

```
Correspondência ()-[r: Has_been_in]->()
Retornar r LIMITE 25
```

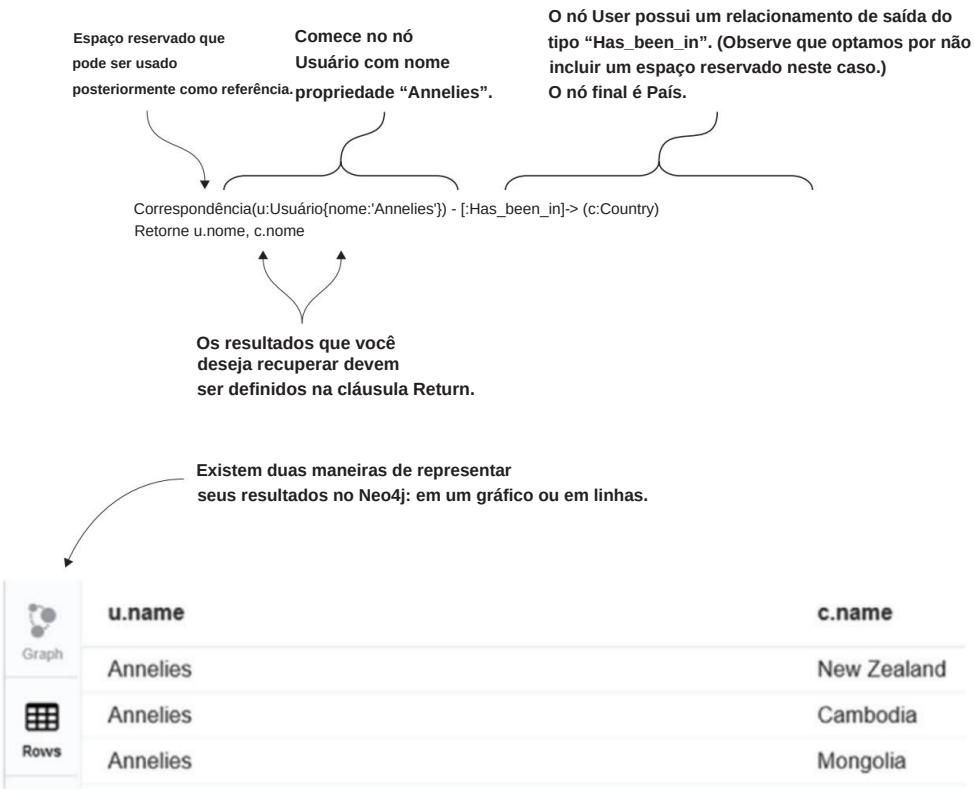


**Figura 7.11 Resultados da pergunta 1: Que países Annelies visitou? Podemos ver os três países em que Annelies esteve, usando a apresentação em linha do Neo4j. A travessia levou apenas 97 milissegundos.**

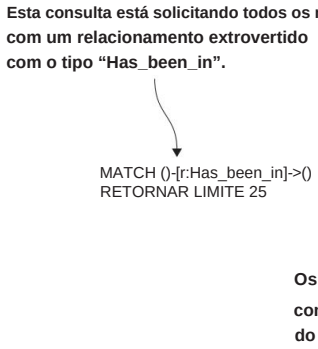


**Figura 7.12 Quem esteve onde? Explicação do acúmulo de consulta.**

Quando executamos esta consulta obtemos a resposta mostrada na figura 7.13.

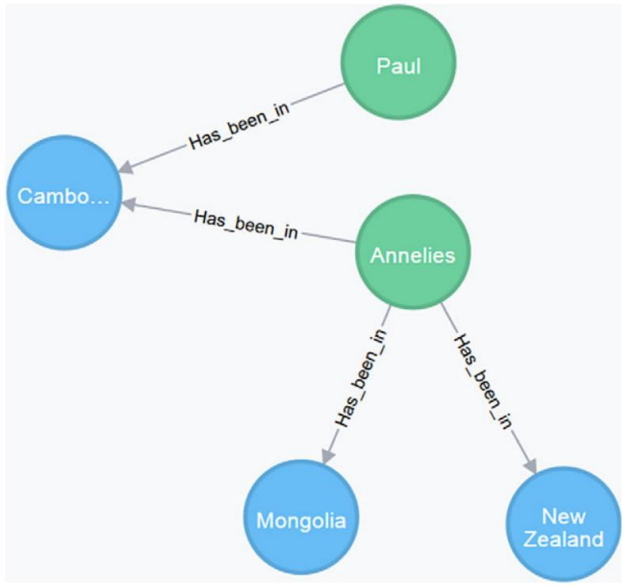


**Figura 7.13 Resultados da questão 2: Quem esteve onde? Os resultados do nosso travessia agora são mostradas no representação gráfica do Neo4j. Agora podemos constatar que Paul, além de Annelies, também esteve no Camboja.**

Na questão 2 optamos por não especificar um nó inicial. Portanto, Cypher irá
a todos os nós presentes no banco de dados para encontrar aqueles com um relacionamento de saída de digite “Has_been_in”. Deve-se evitar não especificar um nó inicial, pois, dependendo do tamanho do seu banco de dados, tal consulta pode levar muito tempo para convergir.

Brincar com os dados para obter o banco de dados gráfico correto também significa muito trabalho. exclusão de dados. Cypher possui uma instrução delete adequada para excluir pequenas quantidades de

dados. A consulta a seguir demonstra como excluir todos os nós e relacionamentos no banco de dados:

```
PARTIDA(n)
Opcional MATCH (n)-[r]-()
Excluir n,r
```

Agora que estamos familiarizados com dados conectados e temos conhecimento básico de como eles são gerenciados em um banco de dados gráfico, podemos dar um passo adiante e examinar aplicações reais e em tempo real de dados conectados. Um gráfico social, por exemplo, pode ser usado para encontrar clusters de nós fortemente conectados dentro das comunidades gráficas. As pessoas de um grupo que não se conhecem podem então ser apresentadas umas às outras. O conceito de busca por nós fortemente conectados, nós que possuem uma quantidade significativa de características em comum, é um conceito amplamente utilizado. Na próxima seção usaremos essa ideia, onde o objetivo será encontrar clusters dentro de uma rede de ingredientes.

## 7.3 Exemplo de dados conectados: um mecanismo de recomendação de receita

Um dos

casos de uso mais populares para bancos de dados gráficos é o desenvolvimento de mecanismos de recomendação. Os mecanismos de recomendação foram amplamente adotados por meio de sua promessa de criar conteúdo relevante. Viver em uma era com tanta abundância de dados pode ser opressor para muitos consumidores. As empresas perceberam a clara necessidade de serem criativas na forma de atrair clientes através de conteúdo personalizado, utilizando assim os pontos fortes dos motores de recomendação.

Em nosso estudo de caso, recomendaremos receitas baseadas nas preferências de pratos dos usuários e em uma rede de ingredientes. Durante a preparação dos dados, usaremos o Elasticsearch para agilizar o processo e permitir mais foco no banco de dados gráfico real. Seu principal objetivo aqui será substituir a lista de ingredientes dos dados “sujos” baixados pelos ingredientes de nossa própria lista “limpa”.

Se você pulou para este capítulo, talvez seja bom ler pelo menos o apêndice A sobre como instalar o Elasticsearch para executá-lo em seu computador. Você sempre pode baixar o índice que usaremos na página de download da Manning para este capítulo e colá-lo em seu diretório de dados local do Elasticsearch se não quiser se preocupar com o estudo de caso do capítulo 6.

Você pode baixar as seguintes informações do site da Manning para este capítulo:

Três arquivos de código .py e suas contrapartes .ipynb ÿ

*Preparação de dados, parte* 1 — fará upload dos dados para o Elasticsearch (como alternativa, você pode colar o índice para download em sua pasta de dados local do Elasticsearch)

ÿ *Preparação de dados, parte* 2 — Moverá os dados do Elasticsearch para o Neo4j. ÿ

Sistema de exploração e recomendação

Três arquivos de dados

- ÿ *Ingredientes* (.txt)—Arquivo de ingredientes autocompilado
- ÿ *Receitas* (.json)—Contém todos os ingredientes
- ÿ *Índice Elasticsearch* (.zip)—Contém o índice “gastronômico” do Elasticsearch que você pode usar para pular a parte 1 da preparação de dados

Agora que temos tudo o que precisamos, vejamos o objetivo da pesquisa e as etapas que precisamos seguir para alcançá-lo.

### 7.3.1 Passo 1: Definir o objetivo da pesquisa

Vejamos o que está por vir quando seguirmos o processo de ciência de dados (figura 7.14).

Nosso objetivo principal é configurar um mecanismo de recomendação que ajude os usuários de um site de culinária a encontrar a receita certa. Um usuário pode gostar de diversas receitas e vamos basear

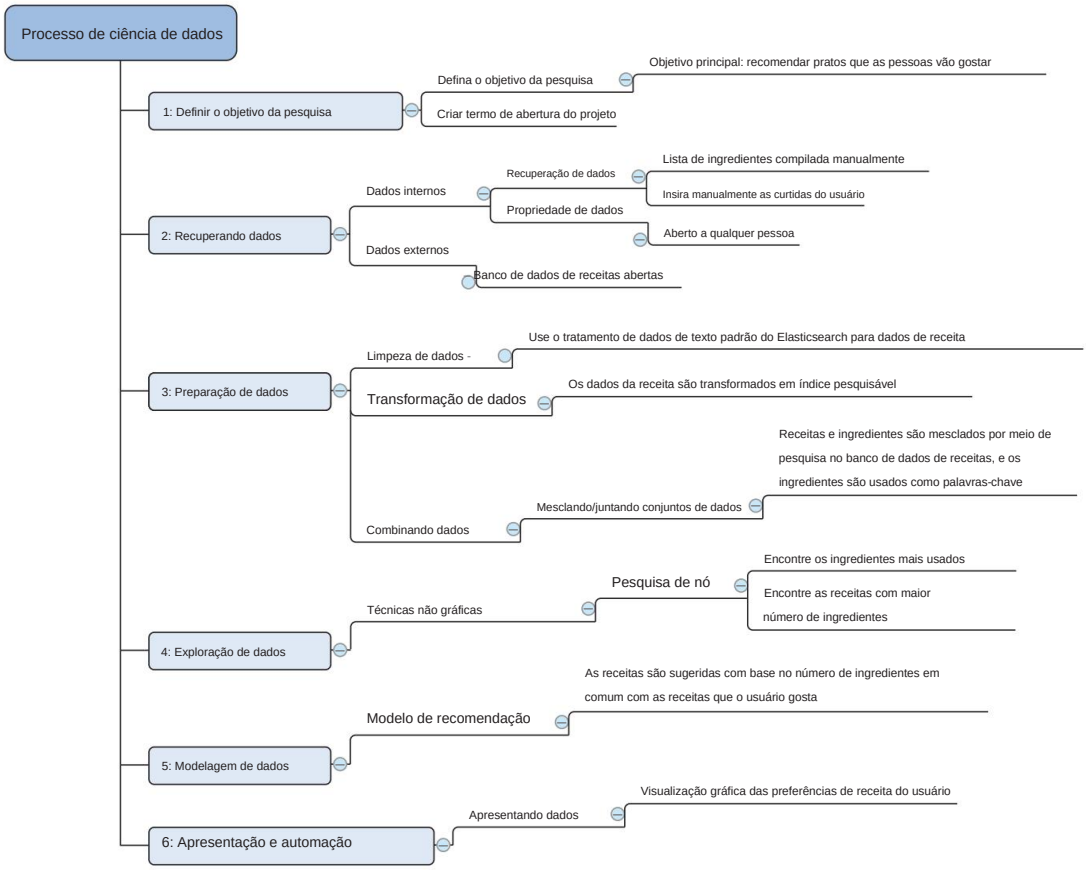


**Figura 7.14 Visão geral do processo de ciência de dados aplicado ao modelo de recomendação de dados conectados**

nossas recomendações de pratos sobre a sobreposição de ingredientes em uma rede de receitas. Esta é uma abordagem simples e intuitiva, mas já produz resultados bastante precisos. Vejamos os três elementos de dados que necessitamos.

### 7.3.2 Etapa 2: Recuperação de dados

Para este exercício, precisamos de três tipos de dados:

- ÿ Receitas e seus respectivos ingredientes ÿ Uma lista de ingredientes distintos que gostamos de modelar
- ÿ Pelo menos um usuário e sua preferência por determinados pratos

Como sempre, podemos dividir isso em dados disponíveis internamente ou criados e dados adquiridos externamente.

- ÿ *Dados internos* – Não temos quaisquer *preferências de usuário* ou ingredientes disponíveis, mas estes são a menor parte de nossos dados e são facilmente criados. Algumas preferências inseridas manualmente devem ser suficientes para criar uma recomendação. O usuário obtém resultados mais interessantes e precisos quanto mais feedback ele dá. Inseriremos as preferências do usuário posteriormente no estudo de caso. Uma *lista de ingredientes* pode ser compilada manualmente e permanecerá relevante por muitos anos, então sinta-se à vontade para usar a lista no material para download para qualquer finalidade, comercial ou não. ÿ *Dados externos* – As receitas são uma questão diferente. Existem milhares de ingredientes, mas estes podem ser combinados em milhões de pratos. Estamos com sorte, entretanto, porque uma lista bem grande está disponível gratuitamente em https://github.com/fictivekin/openrecip Muito obrigado à Fictive Kin por este valioso conjunto de dados com mais de cem mil receitas. Claro que há duplicatas aqui, mas elas não prejudicarão tanto nosso caso de uso.

Agora temos dois arquivos de dados à nossa disposição: uma lista de mais de 800 ingredientes (ingredients.txt) e mais de cem mil receitas no arquivo Recipes.json. Uma amostra da lista de ingredientes pode ser vista na listagem a seguir.

**Listagem 7.2 Exemplo de arquivo de texto de lista de ingredientes**

```
Amarrado
Macarrão De Ovo
Borboletas
Fettuccine
Fusilli
Lasanha
Macarrão espaguete achatado
Macarrão
Orzo
```

O arquivo JSON “openrecipes” contém mais de cem mil receitas com múltiplas propriedades, como data de publicação, local de origem, tempo de preparação, descrição,

e assim por diante. Estamos interessados apenas no nome e na lista de ingredientes. Um exemplo de receita é mostrado na listagem a seguir.

**Listagem 7.3 Um exemplo de receita JSON**

```
{ "_id" :          { "$oid": "5160756b96cc62079cc2db15" },
      "name" : "Biscoitos e molho de salsicha", "ingredients" :
      "Biscoitos\n3 xícaras de farinha multiuso\n2 colheres de sopa de fermento em pó\n1/2 colher de chá de
        sal\n1-1/2 palito (3 /4 xícara) de manteiga fria, cortada em pedaços\n1-1/4 xícara de leitelho\n MOLHO DE
        SALSICHA\n1 quilo de salsicha de café da manhã, quente ou suave\n1/3 xícara de farinha de trigo\n4
        xícaras de leite integral\n1/2 colher de chá Sal temperado\n2 colheres de chá de pimenta preta, mais a
        gosto", "url" : "http://thepioneerwoman.com/cooking/2013/03/drop-biscuits-and-
      sausage-gravy/", "image" : "http ://static.thepioneerwoman.com/cooking/files/2013/03/


        bisgrav.jpg", { "$date":
                  1365276011104 }, "ts": "cookTime": "PT30M", "source":
      "thepioneerwoman", "recipeYield": "12",
      "datePublished": "2013-03 -11", "prepTime" : "PT10M",
      "description" : "No final da tarde de sábado,
      depois que o Homem de Marlboro voltou para casa com as meninas
      jogadoras de futebol, e eu voltei para casa
      com o..."

}
```

Como estamos lidando com dados de texto aqui, o problema é duplo: primeiro, preparar o dados textuais conforme descrito no capítulo mineração de texto. Então, uma vez que os dados estejam completamente limpos, eles podem ser usados para produzir recomendações de receitas baseadas em uma rede de ingredientes. Este capítulo não se concentra na preparação de dados de texto porque isso está descrito em outro lugar, então nos permitiremos o luxo de um atalho durante o próxima preparação de dados.

### 7.3.3 Etapa 3: Preparação de dados

Agora temos dois arquivos de dados à nossa disposição e precisamos combiná-los em um banco de dados gráfico. Os dados de receitas “sujas” representam um problema que podemos resolver usando nossa lista de ingredientes limpos e o uso do mecanismo de busca e do banco de dados NoSQL Elasticsearch. Já contamos com o Elasticsearch em um capítulo anterior e agora ele irá limpe os dados da receita implicitamente para nós ao criar um índice. Podemos então pesquisar esses dados para vincular cada ingrediente a cada receita em que ele ocorre. Poderíamos limpar o dados de texto usando Python puro, como fizemos no capítulo de mineração de texto, mas isso mostra que é é bom estar atento aos pontos fortes de cada banco de dados NoSQL ; não se prenda a um tecnologia única, mas usá-las em conjunto para o benefício do projeto.

Vamos começar inserindo os dados da nossa receita no Elasticsearch. Se você não entende o que está acontecendo, verifique novamente o estudo de caso do capítulo 6 e ele deve ficar claro. Certifique-se de ativar sua instância local do Elasticsearch e ativar um Ambiente Python com o módulo Elasticsearch instalado antes de executar o código

trecho na listagem a seguir. É recomendado não executar este código “como está” no Ipy-thon (ou Jupyter) porque ele imprime todas as chaves de receita na tela e no seu navegador pode lidar apenas com uma certa quantidade de saída. Desative as instruções de impressão ou execute outro IDE Python. O código neste snippet pode ser encontrado em “Preparação de Dados Parte 1.py”.

**Listagem 7.4 Importando dados de receita para o Elasticsearch**

```
da importação do elasticsearch Importação do Elasticsearch json

cliente = Elasticsearch () indexName =
"gastronômico"
docType = 'receitas'

cliente.indices.create(index=nomedoíndice)

nome_do_arquivo = 'C:/Usuários/Usuário/Downloads/recipes.json'

RecipeMapping =
        { 'propriedades': {
            'nome': {'tipo': 'string'}, 'ingredientes': {'tipo':
            'string'}
        }
    }

client.indices.put_mapping(index=indexName,doc_type=docType,body=recipeMapping )

com open(file_name, encoding="utf8") como data_file: RecipeData =
    json.load(data_file)

para receita em RecipeData:
    imprimir receita.keys()
    imprimir receita['_id'].keys()
    cliente.index(index=nomedoíndice,
        doc_type=docType,id = receita['_id']['$oid'],
        body={"nome": receita['nome'], "ingredientes":receita['ingredientes']})
```

**Importe módulos.**

**Cliente Elasticsearch usado para comunicação com banco de dados.**

**Criar índice.**

**Localização do arquivo de receita JSON: altere para corresponder à sua configuração!**

**Mapeamento para tipo de documento “receita” do Elasticsearch.**

**Carregue o arquivo de receita JSON na memória. Outra maneira de fazer isso seria: RecipeData = []**

```
com open(nome_do_arquivo) como f:
    para linha em f:
        receitaData.append(json.loads(linha))
```

**Receitas de índice. Apenas nome e ingredientes são importantes para nosso caso de uso. Caso um ocorre um problema de tempo limite, é possível aumentar o atraso de tempo limite especificando, por por exemplo, timeout=30 como argumento.**

Se tudo corresse bem, agora temos um índice Elasticsearch com o nome “gastronômico” preenchido por milhares de receitas. Observe que permitimos duplicatas do mesmo receita, não atribuindo o nome da receita como chave do documento. Se, por

Por exemplo, uma receita é chamada “lasanha”, então pode ser uma lasanha de salmão, lasanha de carne, lasanha de frango ou qualquer outro tipo. Nenhuma receita é selecionada como protótipo de lasanha; todos eles são carregados no Elasticsearch com o mesmo nome: “lasanha”. Isto é um escolha, então sinta-se à vontade para decidir o contrário. Terá um impacto significativo, como veremos mais tarde. A porta está agora aberta para um upload sistemático para nosso banco de dados gráfico local. Certifique-se de que sua instância de banco de dados gráfico local esteja ativada ao aplicar o código a seguir. Nosso nome de usuário para este banco de dados é o Neo4j padrão e a senha é Neo4ja; certifique-se de ajustar isso para sua configuração local. Para isso também precisaremos de um Biblioteca Python específica do Neo4j chamada py2neo. Se ainda não o fez, agora seria o hora de instalá-lo em seu ambiente virtual usando pip install py2neo ou conda instale o py2neo ao usar o Anaconda. Novamente, esteja ciente de que este código irá travar seu navegador quando executado diretamente no Ipython ou Júpiter. O código nesta listagem pode ser encontrado em “Preparação de Dados Parte 2.py”.

**Listagem 7.5 Usando o índice Elasticsearch para preencher o banco de dados gráfico**

```
da importação do elasticsearch Elasticsearch
de py2neo importar gráfico, autenticar, nó, relacionamento

cliente = Elasticsearch()
indexName = "gastronômico" docType =
'receitas'

authenticate("localhost:7474", "usuário", "senha") graph_db = Graph("http://
localhost:7474/db/data/")

nome do arquivo = 'C:/Users/Gebruiker/Downloads/ingredients.txt' ingredientes =[]

com open (nome do arquivo) como f:
    para linha em f:
        ingredientes.append(line.strip())

imprimir ingredientes

número do ingrediente = 0
total geral = 0
para ingrediente em ingredientes:

    tentar:
        IngredientNode = graph_db.merge_one("Ingrediente","Nome",ingrediente)
    exceto:
        continuar

    número do ingrediente +=1
    corpo de pesquisa =
        { "tamanho": 99999999,
        "consulta": {
            "frase_correspondência":
                {
                    "ingredientes":{
```

**Importar módulos**

**Cliente Elasticsearch usado para se comunicar com o banco de dados**

**Autenticar com seu próprio nome de usuário e senha**

**Gráfico base de dados entidade**

**O arquivo de texto de ingredientes é carregado na memória**

**Tira por causa do /n que você obtém ao ler o .txt**

**Percorra os ingredientes e busque Resultado do Elasticsearch**

**Criar nó no gráfico banco de dados para o atual ingrediente**

**Correspondência de frase usada, pois alguns ingredientes consistem em várias palavras**

```
                                    "consulta": ingrediente,
                                }
                            }
                        }
                    }
                resultado = client.search(index=indexName,doc_type=docType,body=searchbody)

                imprimir ingrediente
                imprimir número do ingrediente
                print "total: " + str(resultado['hits']['total'])

                total geral = total geral + resultado['hits']['total']
                imprima "total geral:"                  + str(total geral)

                para receita em resultado['hits']['hits']:

                    tentar:
                        ReceitaNode =
                graph_db.merge_one("Receita","Nome",receita['_source']['nome'])
                        NodesRelationship = Relacionamento(RecipeNode, "Contém", IngredientNode)

                        graph_db.create_unique(NodesRelationship) print "adicionado: " +
                        receita['_source']['nome'] + ingrediente                    "contém"          +

                    exceto:
                        continuar

                imprimir "**********************************"
```

**Percorra as receitas encontradas para este ingrediente específico**

**Criar relação entre isso receita e ingrediente**

**Crie um nó para cada receita que não é já está no gráfico base de dados**

Ótimo, agora somos orgulhosos proprietários de um banco de dados gráfico repleto de receitas! É tempo de exploração de dados conectada.

### 7.3.4 Etapa 4: Exploração de dados

Agora que temos nossos dados onde queremos, podemos explorá-los manualmente usando o Interface Neo4j em http://localhost:7474/browser/.

Nada impede você de executar seu código Cypher neste ambiente, mas Cypher também pode ser executado através da biblioteca py2neo. Uma questão interessante que podemos colocar é quais ingredientes estão ocorrendo mais em todas as receitas? O que temos maior probabilidade de obter em nosso sistema digestivo se selecionássemos e comêssemos aleatoriamente pratos desse banco de dados?

```
de py2neo importar gráfico, autenticar, nó, relacionamento
autenticar("localhost:7474", "usuário", "senha")
gráfico_db = Gráfico("http://localhost:7474/db/data/")graph_db.cypher.execute("
    MATCH (REC:Receita)-[r:Contém]->(ING:Ingrediente) COM ING, contagem(r) AS num
    RETURN ING.Name como Nome, num ORDER BY num DESC LIMIT 10;")
```

A consulta é criada no Cypher e diz: para todas as receitas e seus ingredientes, conte o número de relações por ingrediente e devolva os dez ingredientes com mais relações e suas respectivas contagens. Os resultados são mostrados na figura 7.15.

| | Name | num |
|---|---|---|
| 1 | Salt | 53885 |
| 2 | Oil | 42585 |
| 3 | Sugar | 38519 |
| 4 | Pepper | 38118 |
| 5 | Butter | 35610 |
| 6 | Garlic | 29879 |
| 7 | Flour | 28175 |
| 8 | Olive Oil | 25979 |
| 9 | Onion | 24888 |
| 10 | Cloves | 22832 |

**Figura 7.15 Os 10 principais ingredientes que ocorrem na maioria das receitas**

A maior parte da lista dos 10 primeiros na figura 7.15 não deveria vir tão uma surpresa. Com o sal orgulhosamente no topo da nossa lista, nós não deveria ficar chocado ao descobrir doenças vasculares como o número é o único assassino na maioria dos países ocidentais. Outro interesse-

A pergunta que vem à mente agora é de um ponto diferente perspectiva: quais receitas requerem mais ingredientes?

```
de py2neo importar gráfico, nó, relacionamento
gráfico_db = Gráfico("http://neo4j:neo4ja@localhost:7474/db/data/")
gráfico_db.cypher.execute("
    MATCH (REC:Receita)-[r:Contém]->(ING:Ingrediente) COM REC, contagem(r) AS num
    RETURN REC.Name como Nome, num ORDER BY num DESC LIMIT 10;")
```

A consulta é quase a mesma de antes, mas em vez de retornar os ingredientes, exija as receitas. O resultado é a figura 7.16.

| | Name | num |
|---|---|---|
| 1 | Spaghetti Bolognese | 59 |
| 2 | Chicken Tortilla Soup | 56 |
| 3 | Kedgeree | 55 |
| 4 | Butternut Squash Soup | 54 |
| 5 | Hearty Beef Stew | 53 |
| 6 | Chicken Tikka Masala | 52 |
| 7 | Fish Tacos | 52 |
| 8 | Cooking For Others: 25 Years of Jor, 1 of BGSK | 51 |
| 9 | hibernation fare | 50 |
| 10 | Gazpacho | 50 |

**Figura 7.16 Os 10 melhores pratos que podem ser criados com a maior diversidade de ingredientes**

Agora, esta pode ser uma visão surpreendente. Espaguete à bolonhesa dificilmente parece o tipo de prato que exigiria 59 ingredientes. Vamos dar uma olhada mais de perto nos ingredientes listado para espaguete à bolonhesa.

```
de py2neo importar gráfico, nó, relacionamento
gráfico_db = Gráfico("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("MATCH (REC1:Receita{Nome:'Espaguete à Bolonhesa'})-
    [r:Contém]->(ING:Ingrediente) RETURN REC1.Nome, ING.Nome;")
```

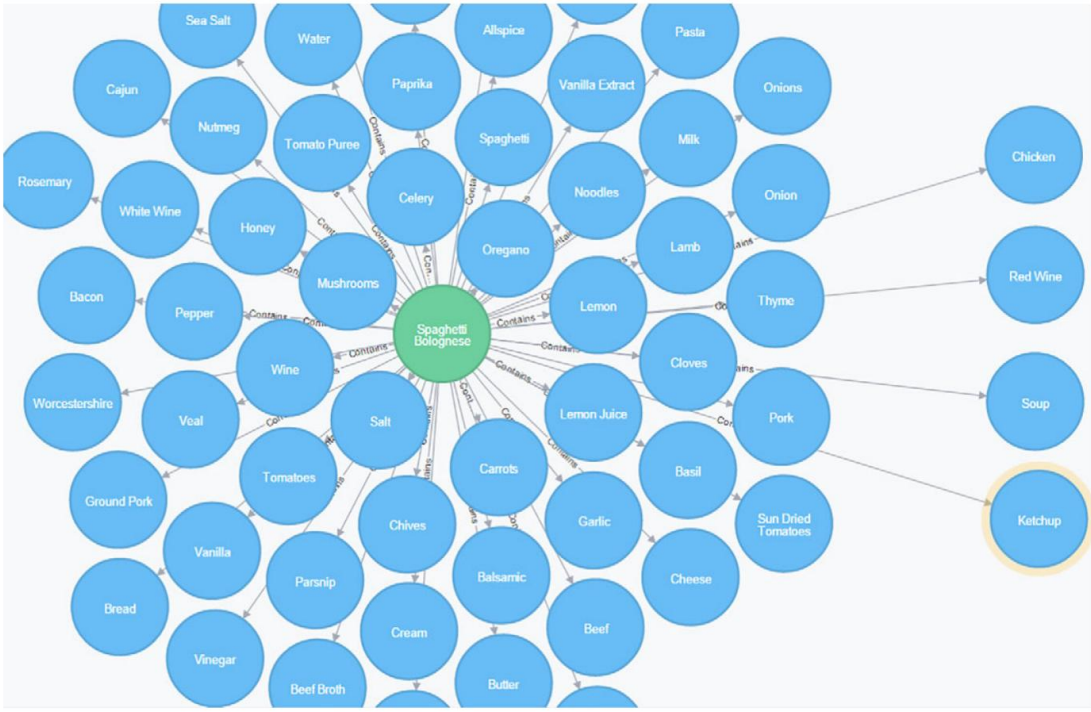


**Figura 7.17 Possíveis ingredientes do espaguete à bolonhesa**

A consulta Cypher apenas lista os ingredientes vinculados ao espaguete à bolonhesa. A Figura 7.17 mostra o resultado na interface web do Neo4j.

Vamos nos lembrar da observação que fizemos ao indexar os dados no Elastic-search. Uma rápida pesquisa no Elasticsearch sobre Spaghetti Bolognese nos mostra que isso ocorre várias vezes, e todas essas instâncias foram usadas para vincular ingredientes ao Spaghetti Bolonhesa como receita. Não precisamos olhar para o espaguete à bolonhesa como uma receita única, mas mais como uma coleção de maneiras pelas quais as pessoas criam seu próprio “espaguete à bolonhesa”. Isso é uma maneira interessante de analisar esses dados. As pessoas podem criar sua versão do prato com ketchup, vinho tinto e frango ou podem até adicionar sopa.

Sendo o “Espaguete à Bolonhesa” um prato tão aberto à interpretação, não é de admirar que muitas pessoas adoram.

A história do espaguete à bolonhesa foi uma distração interessante, mas não o que viemos para. É hora de recomendar pratos ao nosso gourmet “Ragnar”.

### 7.3.5 Etapa 5: Modelagem de dados

Com o nosso conhecimento dos dados ligeiramente enriquecido, chegamos ao objetivo deste exercício: as recomendações.

Para isso apresentamos um usuário que chamamos de “Ragnar”, que gosta de alguns pratos. Esse novas informações precisam ser absorvidas pelo nosso banco de dados gráfico antes que possamos esperar para sugerir novos pratos. Portanto, vamos agora criar o nó de usuário do Ragnar com alguns preferências de receita.

**Listagem 7.6 Criando um nó de usuário que gosta de certas receitas no banco de dados gráfico Neo4j**

```
de py2neo importar gráfico, nó, relacionamento

gráfico_db = Gráfico("http://neo4j:neo4ja@localhost:7474/db/data/")

UserRef = graph_db.merge_one("Usuário","Nome","Ragnar")

RecipeRef = graph_db.find_one("Receita",property_key="Nome", property_value="Espaguete à
        Bolonhesa")
NodesRelationship = Relationship(UserRef, "Likes", RecipeRef) graph_db.create_unique(NodesRelationship)
#Comprometa seu like no banco de dados

graph_db.create_unique(Relationship(UserRef, "Likes",
        graph_db.find_one("Recipe",property_key="Name", property_value="Sopa de
        tomate assado com pequenas almôndegas e arroz")))

graph_db.create_unique(Relacionamento(UserRef, "Curtir",
        graph_db.find_one("Receita",property_key="Nome", property_value="Moussaka")))

graph_db.create_unique(Relationship(UserRef, "Likes",
        graph_db.find_one("Receita",property_key="Nome", property_value="Chipolata
        e fritada de cebolinha")))
graph_db.create_unique(Relationship(UserRef, "Curtidas",
        graph_db.find_one("Receita",property_key="Nome", property_value="Almôndegas
        com molho de tomate")))
graph_db.create_unique(Relationship(UserRef, "Likes",
        graph_db.find_one("Receita",property_key="Nome", property_value="Macarrão
        com queijo")))
graph_db.create_unique(Relationship(UserRef, "Curtidas",
        graph_db.find_one("Receita",property_key="Nome", property_value="Bife
        apimentado")))
```

**Importar módulos**

**Faça objeto de conexão de banco de dados gráfico**

**Criar novo usuário chamado "Ragnar"**

**Ragnar gosta Espaguete bolonhesa**

**Encontrar receita pelo nome de Espaguete bolonhesa**

**Crie uma curtida relação entre Ragnar e a espaguete**

**Repita o mesmo processo das linhas acima, mas para vários outros pratos**

Na listagem 7.6, nosso conhecedor de comida Ragnar é adicionado ao banco de dados junto com sua preferência por alguns pratos. Se selecionarmos Ragnar na interface do Neo4j, obtemos a figura 7.18. A consulta Cypher para isso é

```
MATCH (U:Usuário)-[r:Gostos]->(REC:Receita) RETURN U,REC LIMIT 25
```

Não há surpresas na figura 7.18: muitas pessoas gostam de esparguete à bolonhesa, e o nosso também O gastrônomo escandinavo Ragnar.

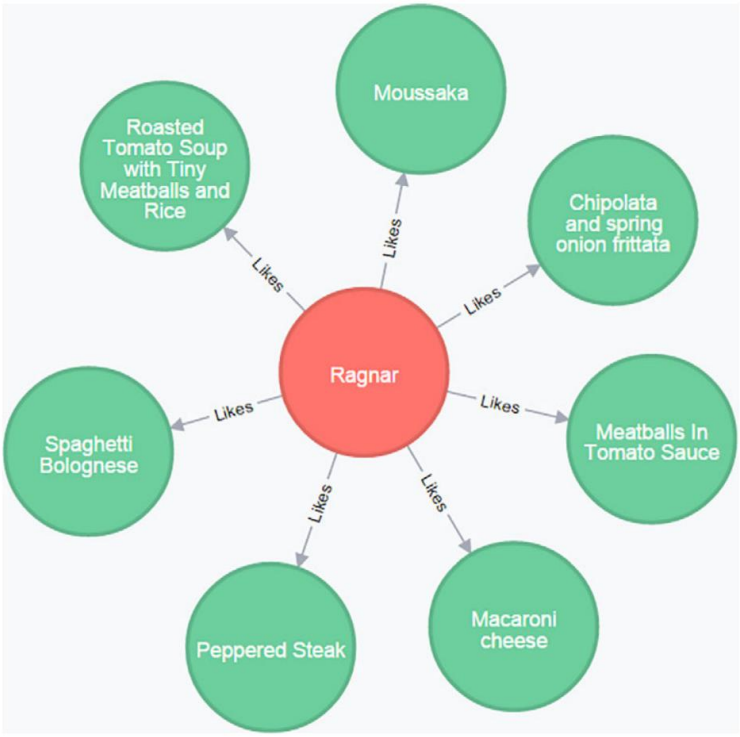


**Figura 7.18 O usuário Ragnar gosta de vários pratos**

Para o mecanismo de recomendação simples que gostamos de construir, tudo o que nos resta fazer é peça ao banco de dados gráfico que nos forneça os pratos mais próximos em termos de ingredientes. Novamente, isso é uma abordagem básica para sistemas de recomendação porque não leva em conta fatores como

- ÿ Não gosto de um ingrediente ou prato.
- ÿ A quantidade de gosto ou desgosto. Uma pontuação de 10 em vez de um binário, goste ou não como poderia fazer a diferença.
- ÿ A quantidade do ingrediente presente no prato.
- ÿ O limite para que um determinado ingrediente se torne aparente no seu sabor. Certo ingredientes, como pimenta picante, representarão um impacto maior por um menor dose do que outros ingredientes fariam.
- ÿ Alergias alimentares. Embora isso seja implicitamente modelado no gosto ou desgosto de pratos com determinados ingredientes, uma alergia alimentar pode ser tão importante que um único erro pode ser fatal. Evitar alergénios deverá substituir todo o sistema de recomendações.
- ÿ Muitas outras coisas para você refletir.

Pode ser uma surpresa, mas um único comando Cypher será suficiente.

```
de py2neo importar gráfico, nó, relacionamento
gráfico_db = Gráfico("http://neo4j:neo4ja@localhost:7474/db/data/")
gráfico_db.cypher.execute("
    MATCH (USR1:Usuário{Nome:'Ragnar'})-[I1:Curtidas]->(REC1:Receita),
          (REC1)-[c1:Contém]->(ING1:Ingrediente)
      COM ING1,REC1 MATCH (REC2:Receita)-[c2:Contém]->(ING1:Ingrediente)
      ONDE REC1 <> REC2
    RETURN REC2.Name,count(ING1) AS IngCount ORDER BY IngCount DESC LIMIT 20;")
```

Primeiro, todas as receitas que Ragnar gosta são coletadas. Então seus ingredientes são usados para busque todos os outros pratos que os compartilham. Os ingredientes são então contados para cada prato conectado e classificado de muitos ingredientes comuns a poucos. Apenas os 20 primeiros os pratos são guardados; isso resulta na tabela da figura 7.19.

| | REC2.Name | IngCount |
|---|---|---|
| 1 | Spaghetti and Meatballs | 104 |
| 2 | Hearty Beef Stew | 91 |
| 3 | Cassoulet | 89 |
| 4 | Lasagne | 88 |
| 5 | Spaghetti & Meatballs | 86 |
| 6 | Good old lasagne | 84 |
| 7 | Beef Wellington | 84 |
| 8 | Braised Short Ribs | 83 |
| 9 | Lasagna | 83 |
| 10 | Italian Wedding Soup | 82 |
| 11 | French Onion Soup | 82 |
| 12 | Coq au vin | 82 |
| 13 | Shepherd's pie | 81 |
| 14 | Great British pork: from head to toe | 81 |
| 15 | Three Meat Cannelloni Bake | 81 |
| 16 | Cioppino | 81 |
| 17 | hibernation fare | 80 |
| 18 | Spaghetti and Meatballs Recipe with Oven Roasted Tomato Sauce | 80 |
| 19 | Braised Lamb Shanks | 80 |
| 20 | Lamb and Eggplant Casserole (Moussaka) | 80 |

**Figura 7.19 Resultado da recomendação da receita; 20 melhores pratos que o usuário pode adorar**

Da figura 7.19 podemos deduzir que é hora de Ragnar experimentar Spaghetti and Meatballs, um prato que ficou imortalmente famoso pela animação da Disney *A Dama e o Vagabundo.* Isso faz parece uma ótima recomendação para alguém que gosta tanto de pratos que contenham massa e almôndegas, mas como podemos ver pela contagem de ingredientes, muitos mais ingredientes voltam esta sugestão. Para nos dar uma pequena ideia do que está por trás disso, podemos mostrar os pratos preferidos, as principais recomendações e alguns de seus ingredientes sobrepostos em uma única imagem gráfica de resumo.

### 7.3.6 Etapa 6: Apresentação

A interface web do Neo4j nos permite executar o modelo e recuperar um gráfico bonito que resume parte da lógica por trás das recomendações. Mostra como os pratos recomendados estão ligados aos pratos preferidos através dos ingredientes. Isto é mostrado na Figura 7.20 e é o resultado final do nosso estudo de caso.

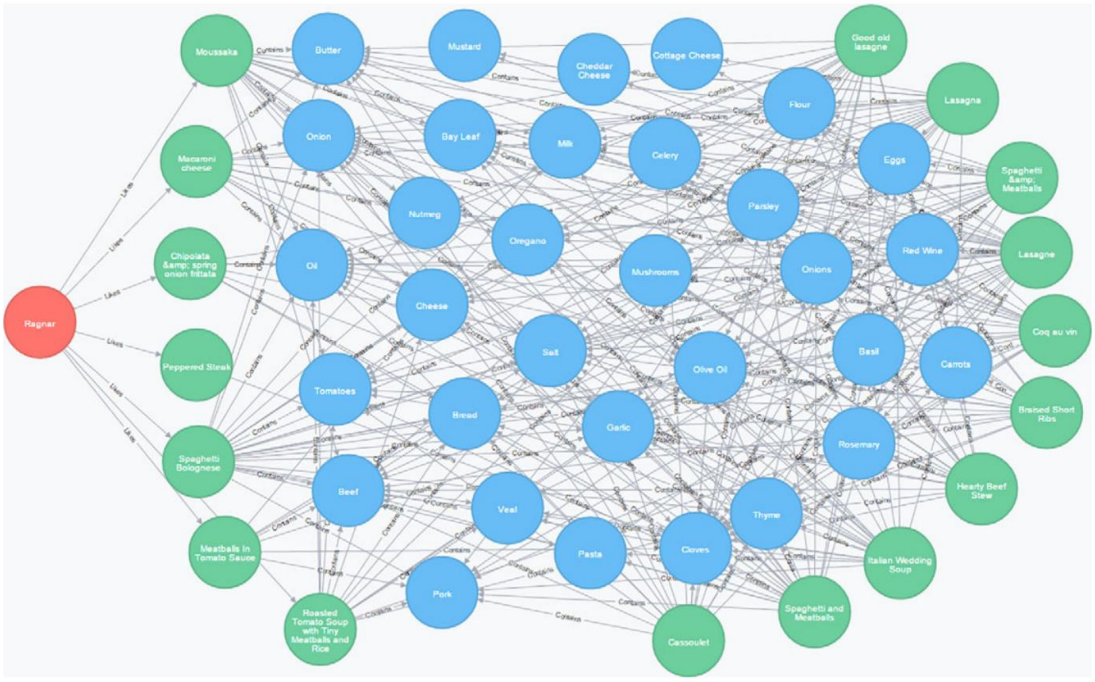


**Figura 7.20 Interconexão entre os pratos preferidos do usuário e os 10 pratos mais recomendados por meio de uma subseleção de seus ingredientes sobrepostos**

Com esta bela imagem gráfica podemos concluir nosso capítulo sabendo que Ragnar tem alguns pratos saborosos pela frente. Não se esqueça de experimentar o sistema de recomendação inserindo suas próprias preferências.

## 7.4 Resumo

Neste capítulo você aprendeu

- ÿ Bancos de dados gráficos são especialmente úteis ao encontrar dados nos quais os relacionamentos entre entidades são tão importantes quanto as próprias entidades. Em comparação com outros bancos de dados NoSQL , eles podem lidar com a maior complexidade mas o mínimo de dados.

ÿ As estruturas de dados gráficos consistem em dois componentes principais: – Nós — são as próprias entidades. Em nosso estudo de caso, são receitas e ingredientes.

– *Arestas—Os* relacionamentos entre entidades. Os relacionamentos, assim como os nós, podem ser de todos os tipos (por exemplo, “contém”, “curtir”, “já visitou”) e podem ter suas próprias propriedades específicas, como nomes, pesos ou outras medidas. ÿ Analisamos o Neo4j, atualmente o banco de dados gráfico mais popular. Para obter instruções sobre como instalá-lo, você pode consultar o apêndice B. Analisamos como adicionar dados ao Neo4j, consultá-los usando Cypher e como acessar sua interface web. ÿ Cypher é a linguagem de consulta específica do banco de dados Neo4j e vimos alguns exemplos. Também o utilizamos no estudo de caso como parte do nosso sistema de recomendação de pratos.

ÿ No estudo de caso do capítulo, usamos o Elasticsearch para limpar um enorme despejo de dados de receitas. Em seguida, convertemos esses dados para um banco de dados Neo4j com receitas e ingredientes. O objetivo do estudo de caso foi recomendar pratos às pessoas com base no interesse previamente demonstrado por outros pratos. Para isso aproveitamos a conexão das receitas através de seus ingredientes. A biblioteca py2neo nos permitiu comunicar com um servidor Neo4j do Python. ÿ Acontece que o banco de dados de gráficos não é útil apenas para implementar um sistema de recomendação, mas também para exploração de dados. Uma das coisas que descobrimos é a diversidade (em termos de ingredientes) das receitas de espaguete à bolonhesa que existem. ÿ Usamos a interface web do Neo4j para criar uma representação visual de como passamos das preferências de pratos às recomendações de pratos por meio dos nós de ingredientes.