

# *Junte-se ao movimento NoSQL*



## **Este capítulo cobre**

- Compreender os bancos de dados NoSQL e por que eles são usados hoje
- Identificando as diferenças entre NoSQL e bancos de dados relacionais
- Definir o princípio ACID e como ele se relaciona com o princípio NoSQL BASE
- Aprender por que o teorema CAP é importante para configuração de banco de dados de vários nós
- Aplicando o processo de ciência de dados a um projeto com o banco de dados NoSQL Elasticsearch

Este capítulo está dividido em duas partes: um início teórico e um final prático.

- Na primeira parte deste capítulo examinaremos os bancos de dados NoSQL em geral e responderemos a estas perguntas: Por que eles existem? Por que não até recentemente? Que tipos existem e por que você deveria se importar?
- Na segunda parte, abordaremos um problema da vida real — diagnóstico de doenças e criação de perfis — usando dados disponíveis gratuitamente, Python e um banco de dados NoSQL .

Sem dúvida você já ouviu falar sobre bancos de dados NoSQL e como eles são usados religiosamente por muitas empresas de alta tecnologia. Mas o que são bancos de dados NoSQL e o que os torna tão diferente dos bancos de dados relacionais ou SQL com os quais você está acostumado? *NoSQL* é a abreviação de *Não Somente linguagem de consulta estruturada*, mas embora seja verdade que os bancos de dados NoSQL podem permitir você os consulta com SQL, não precisa se concentrar no nome real. Muito debate já se enfureceu sobre o nome e se este grupo de novos bancos de dados deveria mesmo têm um nome coletivo. Em vez disso, vamos ver o que eles representam em oposição ao *sistemas de gerenciamento de banco de dados relacional (RDBMS)*. Os bancos de dados tradicionais residem em um único computador ou servidor. Isso costumava ser bom, desde que seus dados não superassem seus servidor, mas já não é o caso de muitas empresas há muito tempo. Com o crescimento da Internet, empresas como Google e Amazon sentiram que estavam presas voltamos a esses bancos de dados de nó único e procuramos alternativas.

Muitas empresas usam bancos de dados NoSQL de nó único, como o MongoDB, porque eles desejam o esquema flexível ou a capacidade de agregar dados hierarquicamente. Aqui estão vários exemplos iniciais:

• A primeira solução NoSQL do Google foi o Google BigTable, que marcou o início do os *bancos de dados colunares*.<sup>1</sup>

• A Amazon criou o Dynamo, um *armazenamento de valores-chave*.<sup>2</sup>

• Mais dois tipos de banco de dados surgiram na busca pelo particionamento: o *documento armazenar* e o *banco de dados gráfico*.

Entraremos em detalhes sobre cada um dos quatro tipos posteriormente neste capítulo.

Observe que, embora o tamanho fosse um fator importante, esses bancos de dados não originam-se unicamente da necessidade de lidar com grandes volumes de dados. Cada V de big data tem influência (volume, variedade, velocidade e às vezes veracidade). Bancos de dados gráficos, para por exemplo, pode lidar com dados de rede. Os entusiastas de bancos de dados gráficos afirmam até que tudo pode ser visto como uma rede. Por exemplo, como você prepara o jantar? Com ingredientes. Esses ingredientes são reunidos para formar o prato e podem ser usados junto com outros ingredientes para formar outros pratos. Visto deste ponto de vista, ingredientes e receitas fazem parte de uma rede. Mas receitas e ingredientes também poderiam ser armazenado em seu banco de dados relacional ou em um armazenamento de documentos; é tudo como você olha para o problema. É aqui que reside a força do NoSQL: a capacidade de analisar um problema de um ponto de vista ângulo diferente, moldando a estrutura de dados para o caso de uso. Como cientista de dados, seu trabalho é encontrar a melhor resposta para qualquer problema. Embora às vezes isso ainda seja mais fácil de Para alcançar usando RDBMS, muitas vezes um banco de dados NoSQL específico oferece uma abordagem melhor.

Os bancos de dados relacionais estão fadados a desaparecer nas empresas com big data porque da necessidade de particionamento? Não, plataformas NewSQL (não confundir com NoSQL) são a resposta do RDBMS à necessidade de configuração de cluster. Os bancos de dados NewSQL seguem o modelo relacional, mas são capazes de ser divididos em um cluster distribuído como NoSQL

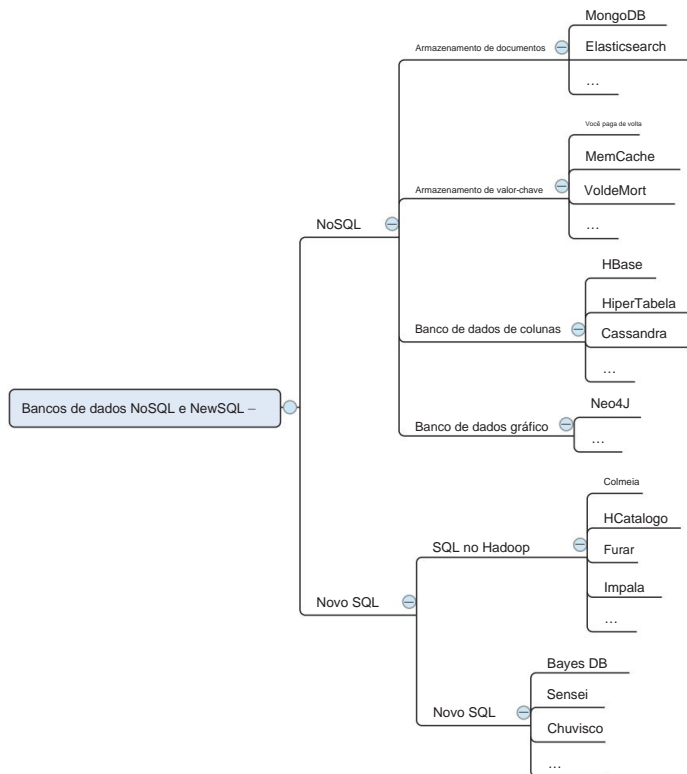
---

<sup>1</sup> Consulte <http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>.

<sup>2</sup> Consulte <http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>.

bancos de dados. Não é o fim dos bancos de dados relacionais e certamente não é o fim do SQL, já que plataformas como o Hive traduzem SQL em trabalhos MapReduce para Hadoop. Além disso, nem toda empresa precisa de big data; muitos se dão bem com bancos de dados pequenos e os bancos de dados relacionais tradicionais são perfeitos para isso.

Se você observar o mapa mental de big data mostrado na Figura 6.1, verá quatro tipos de bancos de dados NoSQL .



**Figura 6.1 Bancos de dados NoSQL e NewSQL**

Esses quatro tipos são armazenamento de documentos, armazenamento de valores-chave, banco de dados gráfico e banco de dados de colunas. O mapa mental também inclui os bancos de dados relacionais particionados NewSQL . No futuro, esta grande divisão entre NoSQL e NewSQL se tornará obsoleta porque cada tipo de banco de dados terá seu próprio foco, ao mesmo tempo que combina elementos de bancos de dados NoSQL e NewSQL . As linhas estão lentamente se confundindo à medida que os tipos de RDBMS obtêm recursos NoSQL , como a indexação orientada a colunas vista em bancos de dados colunares. Mas, por enquanto, é uma boa maneira de mostrar que os antigos bancos de dados relacionais ultrapassaram sua configuração de nó único, enquanto outros tipos de bancos de dados estão surgindo sob o denominador NoSQL .

Vejamos o que o NoSQL traz para a mesa.

## 6.1 Introdução ao NoSQL

Como você leu, o objetivo dos bancos de dados NoSQL não é apenas oferecer uma maneira de particionar bancos de dados com sucesso em vários nós, mas também apresentar maneiras fundamentalmente diferentes de modelar os dados disponíveis para ajustar sua estrutura ao seu caso de uso. e não como um banco de dados relacional exige que ele seja modelado.

Para ajudá-lo a entender o NoSQL, começaremos examinando os princípios básicos do ACID dos bancos de dados relacionais de servidor único e mostraremos como os bancos de dados NoSQL os reescrevem nos princípios BASE para que funcionem muito melhor de maneira distribuída. Também veremos o teorema CAP , que descreve o principal problema com a distribuição de bancos de dados em vários nós e como os bancos de dados ACID e BASE abordam isso.

### 6.1.1 ACID: o princípio fundamental dos bancos de dados relacionais

Os principais aspectos de um banco de dados relacional tradicional podem ser resumidos pelo conceito ACID:

• *Atomicidade* – O princípio “tudo ou nada”. Se um registro for colocado em um banco de dados, ele será inserido completamente ou não será inserido. Se, por exemplo, ocorrer uma falha de energia no meio de uma ação de gravação do banco de dados, você não terminaria com meio registro; não estaria lá de jeito nenhum.

• *Consistência*—Este importante princípio mantém a integridade dos dados. Nenhuma entrada incluída no banco de dados entrará em conflito com regras predefinidas, como a falta de um campo obrigatório ou um campo numérico em vez de texto. • *Isolamento* — Quando algo é alterado no banco de dados, nada pode acontecer exatamente nesses mesmos dados, exatamente no mesmo momento. Em vez disso, as ações acontecem em série com outras alterações. O isolamento é uma escala que vai do baixo isolamento ao alto isolamento. Nessa escala, os bancos de dados tradicionais estão no extremo do “alto isolamento”.

Um exemplo de baixo isolamento seria o Google Docs: várias pessoas podem escrever em um documento exatamente ao mesmo tempo e ver as alterações umas das outras acontecendo instantaneamente. Um documento Word tradicional, no outro extremo do espectro, possui alto isolamento; ele está bloqueado para edição pelo primeiro usuário que o abrir. A segunda pessoa que abre o documento pode visualizar sua última versão salva, mas não consegue ver as alterações não salvas ou editar o documento sem primeiro salvá-lo como uma cópia. Assim, depois que alguém o abre, a versão mais atualizada fica completamente isolada de qualquer pessoa, exceto do editor que bloqueou o documento. • *Durabilidade* —Se os dados entraram no banco de dados, eles deverão

sobreviver permanentemente.

Danos físicos aos discos rígidos destruirão os registros, mas quedas de energia e falhas de software não deveriam.

ACID se aplica a todos os bancos de dados relacionais e a determinados bancos de dados NoSQL , como o banco de dados gráfico Neo4j. Discutiremos mais detalhadamente os bancos de dados gráficos posteriormente neste capítulo e no capítulo 7. Para a maioria dos outros bancos de dados NoSQL, outro princípio se aplica: BASE. Para entender o BASE e por que ele se aplica à maioria dos bancos de dados NoSQL , precisamos examinar o Teorema CAP .

### 6.1.2 Teorema CAP: o problema com bancos de dados em muitos nós

Depois que um banco de dados é distribuído por diferentes servidores, é difícil seguir o princípio ACID devido à consistência que o ACID promete; o Teorema CAP aponta por que isso se torna problemático. O Teorema CAP afirma que um banco de dados pode ser duas das seguintes coisas, mas nunca todas as três:

• *Tolerante a partições* — o banco de dados pode lidar com uma partição de rede ou falha de rede. • *Disponível* — *contanto* que o nó ao qual você está se conectando esteja funcionando e você possa se conectar a ele, o nó responderá, mesmo que a conexão entre os diferentes nós do banco de dados seja perdida.

• *Consistente*—Não importa a qual nó você se conecta, você sempre verá o valor exato dos mesmos dados.

Para um banco de dados de nó único, é fácil ver como ele está sempre disponível e consistente:

• *Disponível*—*Enquanto* o nó estiver ativo, ele estará disponível. Isso é tudo o que a PAC está disponível. promessas da cidade.

• *Consistente* — *Não há* segundo nó, portanto nada pode ser inconsistente.

As coisas ficam interessantes quando o banco de dados é particionado. Então você precisa escolher entre disponibilidade e consistência, conforme mostrado na figura 6.2.

Tomemos o exemplo de uma loja online com servidor na Europa e servidor nos Estados Unidos, com um único centro de distribuição. Um alemão chamado Fritz e um americano chamado Freddy estão fazendo compras ao mesmo tempo na mesma loja online. Eles veem um item e apenas um ainda está em estoque: uma mesa de centro de bronze em formato de polvo. Ocorre um desastre e a comunicação entre os dois servidores locais é

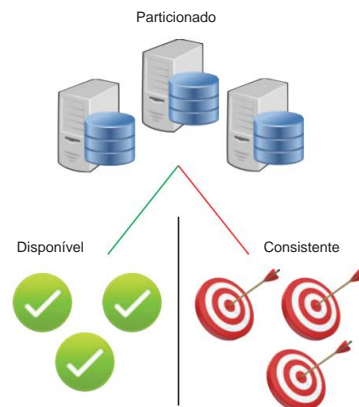
temporariamente para baixo. Se você fosse o dono da loja, teria duas opções:

• *Disponibilidade* — *você* permite que os servidores continuem atendendo os clientes e depois resolve tudo.

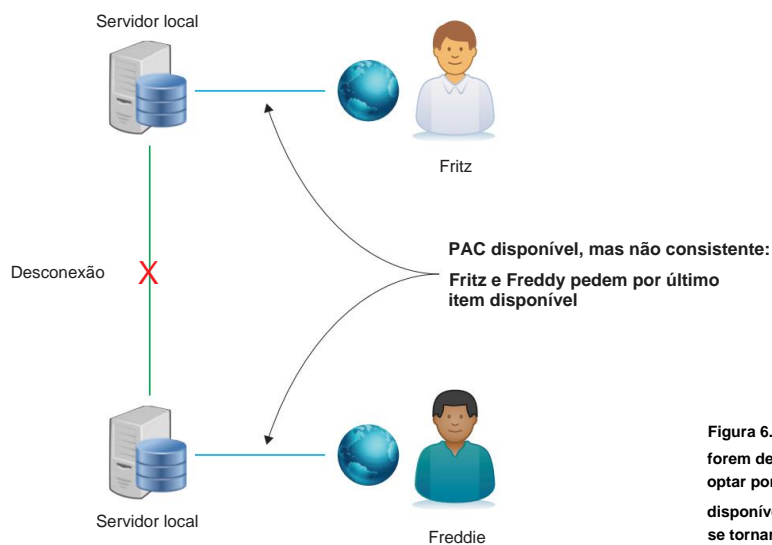
• *Consistência* — *Você* suspende todas as vendas até a comunicação é restabelecida.

No primeiro caso, Fritz e Freddy comprarão a mesa de centro polvo, porque o último número de estoque conhecido para ambos os nós é "um" e ambos os nós podem vendê-la, conforme mostrado na figura 6.3.

Se a mesa de centro for difícil de encontrar, você terá que informar ao Fritz ou ao Freddy que ele não receberá sua mesa na data de entrega prometida ou, pior ainda, ele receberá



**Figura 6.2 Teorema CAP: ao particionar seu banco de dados, você precisa escolher entre disponibilidade e consistência.**



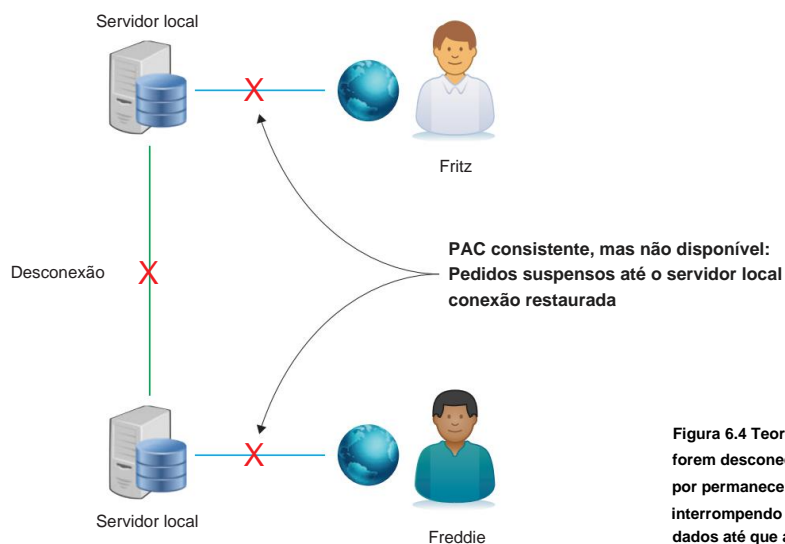
**Figura 6.3 Teorema CAP:** se os nós forem desconectados, você poderá optar por permanecer disponível, mas os dados poderão se tornar inconsistentes.

nunca receba. Como um bom empresário, você pode compensar um deles com um cupom de desconto para uma compra posterior e, afinal, tudo pode ficar bem.

A segunda opção (figura 6.4) envolve colocar as solicitações recebidas em espera temporariamente.

Isso pode ser justo tanto para Fritz quanto para Freddy se depois de cinco minutos a loja virtual estiver aberto para negócios novamente, mas você poderá perder vendas e provavelmente muito mais.

As lojas virtuais tendem a escolher a disponibilidade em vez da consistência, mas não é a escolha ideal



**Figura 6.4 Teorema CAP:** se os nós forem desconectados, você pode optar por permanecer consistente, interrompendo o acesso aos bancos de dados até que as conexões sejam restauradas

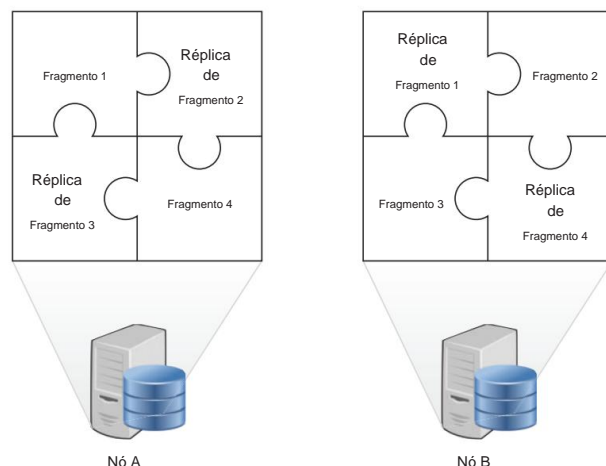
em todos os casos. Veja um festival popular como o Tomorrowland. Os festivais tendem a ter capacidade máxima permitida por razões de segurança. Se você vender mais ingressos do que você permitido porque seus servidores continuaram vendendo durante uma falha de comunicação do nó, você poderia vender o dobro do número permitido quando as comunicações fossem restabelecidas. Nesse caso, pode ser mais sensato buscar consistência e desligar os nós temporariamente. Um festival como o Tomorrowland esgota nas primeiras horas de qualquer maneira, então um pouco de inatividade não vai doer tanto quanto ter que retirar milhares de ingressos.

### 6.1.3 Os princípios BASE dos bancos de dados NoSQL

O RDBMS segue os princípios ACID ; Bancos de dados NoSQL que não seguem ACID, como

os armazenamentos de documentos e armazenamentos de valores-chave, siga BASE. BASE é um conjunto de sons muito mais suaves promessas do banco de dados:

- *Basicamente* disponível – A disponibilidade é garantida no sentido do PAC . Pegando a web Por exemplo, se um nó estiver instalado e funcionando, você poderá continuar comprando. Dependendo de como as coisas estão configuradas, os nós podem assumir o controle de outros nós. Elastic-search, por exemplo, é um mecanismo de pesquisa do tipo documento NoSQL que divide e replica seus dados de tal forma que a falha do nó não significa necessariamente falha de serviço, por meio do processo de *fragmentação*. Cada *fragmento* pode ser visto como uma instância individual do servidor de banco de dados, mas também é capaz de se comunicar com o outros fragmentos para dividir a carga de trabalho da maneira mais eficiente possível (figura 6.5). Vários fragmentos podem estar presentes em um único nó. Se cada fragmento tiver uma réplica em outro nó, a falha do nó é facilmente remediada redividindo o trabalho entre os nós restantes.
- *Estado suave* — O estado de um sistema pode mudar com o tempo. Isto corresponde ao *princípio da consistência eventual*: o sistema pode ter que mudar para tornar os dados



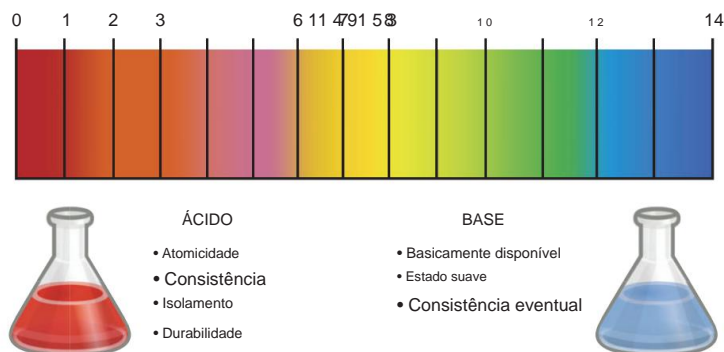
**Figura 6.5** Fragmento: cada fragmento pode funcionar como um fragmento autossuficiente banco de dados, mas eles também trabalham juntos como um todo. O exemplo representa dois nós, cada um contendo quatro fragmentos: dois fragmentos principais e duas réplicas. A falha de um nó é apoiada pelo outro.

consistente novamente. Em um nó os dados podem dizer “A” e no outro pode dizer “B” porque foi adaptado. Mais tarde, na resolução de conflitos, quando a rede estiver novamente online, é possível que o “A” no primeiro nó seja substituído por “B”. Até embora ninguém tenha feito nada para mudar explicitamente “A” para “B”, isso assumirá valor à medida que se torna consistente com o outro nó.

• *Consistência eventual* — O banco de dados se tornará consistente com o tempo. Na web Por exemplo, a tabela é vendida duas vezes, o que resulta em inconsistência de dados. Uma vez a conexão entre os nós individuais for restabelecida, eles se comunicarão e decidirão como resolver o problema. Este conflito pode ser resolvido, por exemplo, em por ordem de chegada ou preferindo o cliente que incorreria o menor custo de transporte. Os bancos de dados vêm com comportamento padrão, mas dado que Se houver uma decisão comercial real a ser tomada aqui, esse comportamento poderá ser substituído. Mesmo que a conexão esteja ativa e funcionando, as latências podem fazer com que os nós tornar-se inconsistentes. Muitas vezes, os produtos são mantidos num carrinho de compras online, mas colocar um item em uma cesta não o bloqueia para outros usuários. Se Fritz vencer Freddy até o botão de finalização da compra, haverá um problema quando Freddy for até Confira. Isso pode ser facilmente explicado ao cliente: ele chegou tarde demais. Mas e se ambos pressionarem o botão de checkout exatamente no mesmo milissegundo e ambos vendas acontecem?

### ÁCIDO versus BASE

Os princípios BASE são um tanto elaborados para se adequar ao ácido e à base da química: um ácido é um fluido com baixo valor de pH. Uma base é o oposto e tem um valor de pH alto. Não entraremos em detalhes de química aqui, mas a figura 6.6 mostra um mnemônico para aqueles familiarizados com os equivalentes químicos de ácido e base.



**Figura 6.6 ACID versus BASE:** bancos de dados relacionais tradicionais versus a maioria dos bancos de dados NoSQL. Os nomes são derivados do conceito químico da escala de pH. Um valor de pH abaixo de 7 é ácido; maior que 7 é uma base. Nesta escala, a água superficial média oscila entre 6,5 e 8,5.



6.1.4 Tipos de banco de dados NoSQL

Como você viu anteriormente, existem quatro grandes tipos de NoSQL : armazenamento de valores-chave, armazenamento de documentos, banco de dados orientado a colunas e banco de dados gráfico. Cada tipo resolve um problema que não pode ser resolvido com bancos de dados relacionais. As implementações reais são frequentemente combinações destes. OrientDB, por exemplo, é um *banco de dados multimodelo*, combinando tipos NoSQL . OrientDB é um banco de dados gráfico onde cada nó é um documento.

Antes de entrarmos nos diferentes bancos de dados NoSQL , vamos dar uma olhada nos bancos de dados relacionais para que você tenha algo com que compará-los. Na modelagem de dados, muitas abordagens são possíveis. Os bancos de dados relacionais geralmente buscam a *normalização*: garantir que cada dado seja armazenado apenas uma vez. A normalização marca sua configuração estrutural. Se, por exemplo, você quiser armazenar dados sobre uma pessoa e seus hobbies, poderá fazê-lo com duas tabelas: uma sobre a pessoa e outra sobre seus hobbies. Como você pode ver na figura 6.7, uma tabela adicional é necessária para vincular hobbies a pessoas devido ao seu *relacionamento muitos-para-muitos*: uma pessoa pode ter vários hobbies e um hobby pode ter muitas pessoas praticando-o.

Um banco de dados relacional em grande escala pode ser composto de muitas entidades e tabelas vinculadas. Agora que você tem algo para comparar o NoSQL , vamos dar uma olhada nos diferentes tipos.

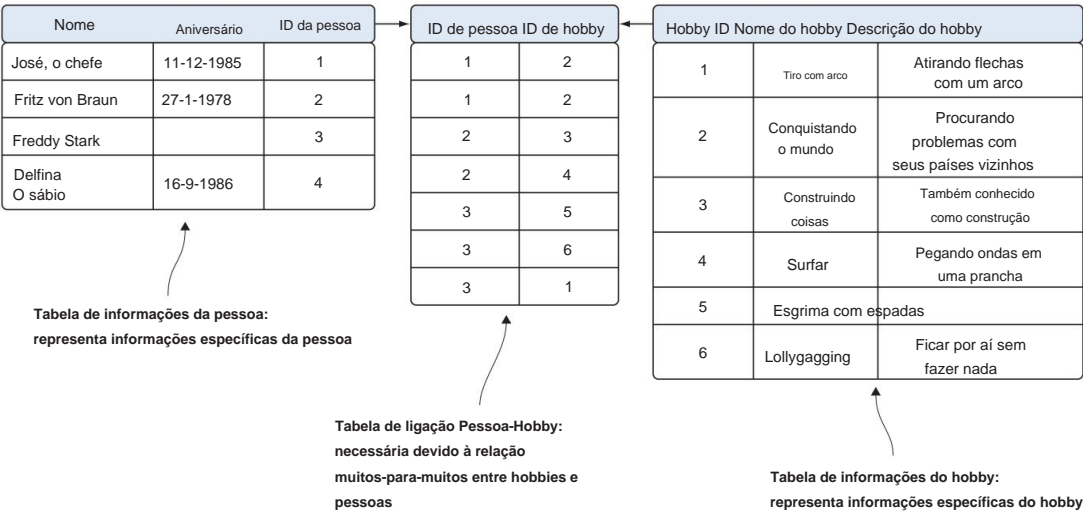


Figura 6.7 Os bancos de dados relacionais buscam a normalização (garantindo que cada dado seja armazenado apenas uma vez). Cada tabela possui identificadores únicos (chaves primárias) que são usados para modelar o relacionamento entre as entidades (tabelas), daí o termo relacional.

BANCO DE DADOS ORIENTADO

**POR COLUNA** Os bancos de dados relacionais tradicionais são orientados por linha, com cada linha tendo um ID de linha e cada campo dentro da linha armazenado juntos em uma tabela. Digamos, por exemplo, que nenhum dado extra sobre hobbies seja armazenado e você tenha apenas uma única tabela para descrever pessoas, como mostra a figura 6.8. Observe como neste cenário você tem uma ligeira desnormalização porque os hobbies podem ser repetidos. Se as informações do hobby forem um ótimo extra, mas não essenciais para o seu caso de uso, adicioná-las como uma lista na coluna Hobbies é uma abordagem aceitável. Mas se a informação não for importante o suficiente para uma tabela separada, ela deveria ser armazenada?

ID da linha	Nome	Aniversário	Hobbies
1	José, o chefe	11-12-1985	Tiro com arco, conquistando o mundo
2	Fritz von Braun	27-1-1978	Construindo coisas, surfando
3	Freddy Stark		Esgrima, pirulito, tiro com arco
4	Delphine Thewiseone 16-9-1986		

Figura 6.8 Layout do banco de dados orientado a linhas. Cada entidade (pessoa) é representada por uma única linha, espalhada por várias colunas.

Cada vez que você procura algo em um banco de dados orientado a linhas, cada linha é verificada, independentemente de quais colunas você precisa. Digamos que você queira apenas uma lista de aniversários em setembro. O banco de dados irá varrer a tabela de cima para baixo e da esquerda para a direita, conforme mostrado na figura 6.9, eventualmente retornando a lista de aniversários.

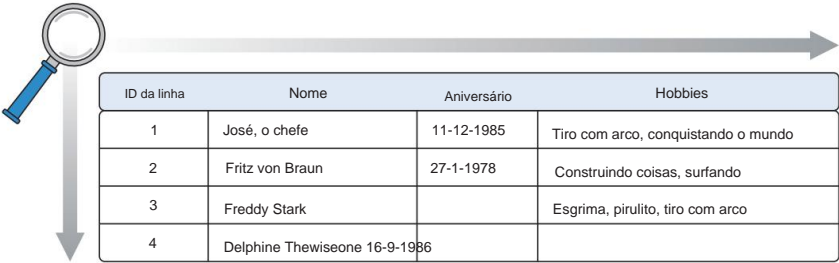


Figura 6.9 Pesquisa orientada a linhas: de cima para baixo e para cada entrada, todas as colunas são levadas para a memória

A indexação dos dados em determinadas colunas pode melhorar significativamente a velocidade de pesquisa, mas a indexação de cada coluna traz sobrecarga extra e o banco de dados ainda verifica todas as colunas.

Os bancos de dados de colunas armazenam cada coluna separadamente, permitindo verificações mais rápidas quando apenas um pequeno número de colunas está envolvido; veja a figura 6.10.

Nome		ID da linha		Aniversário		ID da linha		Hobbies		ID da linha	
José, o chefe		1		11-12-1985		1		Tiro com arco		1, 3	
Fritz von Braun		2		27-1-1978		2		Conquistando o mundo		1	
Freddy Stark		3		16-9-1986		4		Construindo coisas		2	
Delphine Thewiseone		4						Surfar		2	
								Esgrima		3	
								Lollygagging		3	

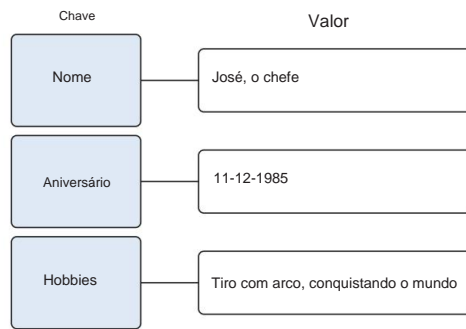
**Figura 6.10** Os bancos de dados orientados a colunas armazenam cada coluna separadamente com os números de linha relacionados. Cada entidade (pessoa) é dividida em várias tabelas.

Esse layout é muito semelhante a um banco de dados orientado a linhas com um índice em cada coluna. Um *índice* de banco de dados é uma estrutura de dados que permite pesquisas rápidas de dados ao custo de espaço de armazenamento e gravações adicionais (atualização de índice). Um índice mapeia o número da linha para os dados, enquanto um banco de dados de colunas mapeia os dados para os números das linhas; dessa forma a contagem fica mais rápida, então é fácil ver quantas pessoas gostam de tiro com arco, por exemplo. Armazenar as colunas separadamente também permite uma compactação otimizada porque há apenas um tipo de dados por tabela.

Quando você deve usar um banco de dados orientado a linhas e quando deve usar um banco de dados orientado a colunas? Em um banco de dados orientado a colunas é fácil adicionar outra coluna porque nenhuma das colunas existentes é afetada por ela. Mas adicionar um registro inteiro requer a adaptação de todas as tabelas. Isso torna o banco de dados orientado a linhas preferível ao banco de dados orientado a colunas para processamento de transações online (OLTP), porque isso implica adicionar ou alterar registros constantemente. O banco de dados orientado a colunas se destaca ao realizar análises e relatórios: somando valores e contando entradas. Um banco de dados orientado a linhas costuma ser o banco de dados operacional preferido para transações reais (como vendas). Os trabalhos em lote noturnos atualizam o banco de dados orientado a colunas, suportando pesquisas e agregações extremamente rápidas usando algoritmos MapReduce para relatórios. Exemplos de lojas de família de colunas são Apache HBase, Cassandra do Facebook, Hypertable e o avô das lojas de colunas largas, Google BigTable.

**LOJAS DE VALOR CHAVE**

Os armazenamentos de valores-chave são os menos complexos dos bancos de dados NoSQL . Eles são, como o nome sugere, uma coleção de pares chave-valor, como mostra a figura 6.11, e essa simplicidade os torna os mais escaláveis dos tipos de banco de dados NoSQL , capazes de armazenar grandes quantidades de dados.



**Figura 6.11** Os armazenamentos de valores-chave armazenam tudo como uma chave e um valor.

O valor em um armazenamento de valores-chave pode ser qualquer coisa: uma string, um número, mas também um valor inteiro. novo conjunto de pares chave-valor encapsulados em um objeto. A Figura 6.12 mostra um pouco mais estrutura complexa de valores-chave. Exemplos de armazenamentos de valores-chave são Redis, Voldemort, Riak, e o Dínamo da Amazon.

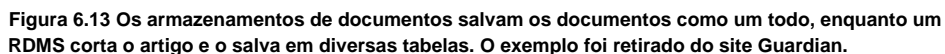
```
{ "internal data": [ { "entities": [
  { "customer": [
    { "id": 1, "name": "Freddy" },
    { "id": 2, "name": "Fritz" }
  ] },
  { "legal entities": [
    { "id": 1, "company": "Maiton" }
  ] }
] }, { "Products": [
  { "furniture": [
    { "id": 1, "name": "Octopus Table", "stock": 1 }
  ] }
] } ] }
```

**Figura 6.12** Estrutura aninhada de valor-chave

### LOJAS DE DOCUMENTOS

Os armazenamentos de documentos estão um passo à frente em complexidade em relação aos armazenamentos de valores-chave: um documento store assume uma determinada estrutura de documento que pode ser especificada com um esquema.

Os armazenamentos de documentos parecem os mais naturais entre os tipos de banco de dados NoSQL porque eles foram projetados para armazenar documentos do dia a dia como estão e permitem consultas e cálculos complexos nessa forma de dados muitas vezes já agregada. Do jeito que as coisas são armazenado em um banco de dados relacional faz sentido do ponto de vista da normalização: tudo deve ser armazenado apenas uma vez e conectado por meio de chaves estrangeiras. Armazenamentos de documentos se preocupam pouco com a normalização, desde que os dados estejam em uma estrutura que faça sentido. A o modelo de dados relacionais nem sempre se adapta bem a determinados casos de negócios. Jornais ou revistas, por exemplo, contêm artigos. Para armazená-los em um banco de dados relacional, você precisa cortá-los primeiro: o texto do artigo vai em uma tabela, o autor e tudo as informações sobre o autor em outro, e comentários ao artigo quando publicado em um site vai para outro. Conforme mostrado na figura 6.13, um artigo de jornal



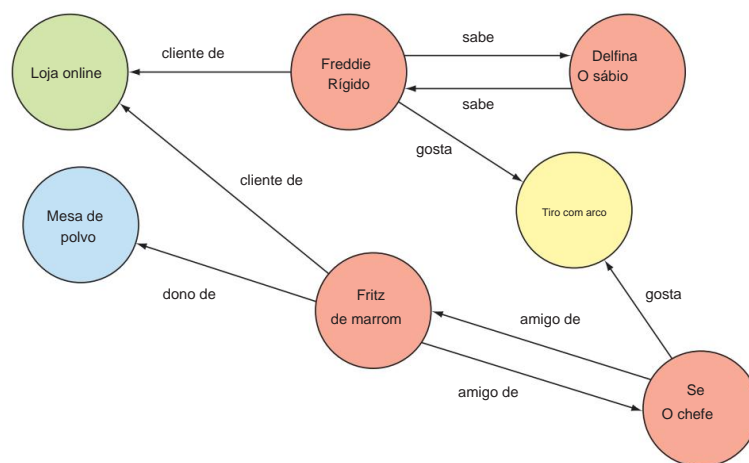
também pode ser armazenado como uma entidade única; isso reduz a carga cognitiva de trabalhar com os dados para aqueles que estão acostumados a ver artigos o tempo todo. Exemplos de armazenamentos de documentos são MongoDB e CouchDB.

#### BANCOS DE DADOS GRÁFICOS

O último grande tipo de banco de dados NoSQL é o mais complexo, voltado para armazenar relações entre entidades de maneira eficiente. Quando os dados estão altamente interconectados, como no caso de redes sociais, citações de artigos científicos ou clusters de ativos de capital, os bancos de dados gráficos são a resposta. Os dados gráficos ou de rede têm dois componentes principais:

• **Nó** — As próprias entidades. Em uma rede social, podem ser pessoas. • **Edge**—O relacionamento entre duas entidades. Essa relação é representada por uma linha e possui propriedades próprias. Uma aresta pode ter uma direção, por exemplo, se a seta indicar quem é o chefe de quem.

Os gráficos podem se tornar incrivelmente complexos devido a tipos de relações e entidades suficientes. A Figura 6.14 já mostra essa complexidade com apenas um número limitado de entidades. Bancos de dados gráficos como o Neo4j também afirmam apoiar o ACID, enquanto os armazenamentos de documentos e valores-chave aderem ao BASE.



**Figura 6.14** Exemplo de dados gráficos com quatro tipos de entidades (pessoa, hobby, empresa e móveis) e suas relações sem arestas extras ou informações de nós

As possibilidades são infinitas e, como o mundo está se tornando cada vez mais interconectado, os bancos de dados gráficos provavelmente ganharão terreno sobre os outros tipos, incluindo o ainda dominante banco de dados relacional. Uma classificação dos bancos de dados mais populares e como eles estão progredindo pode ser encontrada em <http://db-engines.com/en/ranking>.

257 systems in ranking, March 2015

Rank			DBMS	Database Model	Score		
Mar 2015	Feb 2015	Mar 2014			Mar 2015	Feb 2015	Mar 2014
1.	1.	1.	Oracle	Relational DBMS	1469.09	+29.37	-22.71
2.	2.	2.	MySQL	Relational DBMS	1261.09	-11.36	-29.12
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1164.80	-12.68	-40.48
4.	4.	↑ 5.	MongoDB 🟡	Document store	275.01	+7.77	+75.03
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	264.44	+2.10	+29.38
6.	6.	6.	DB2	Relational DBMS	198.85	-3.57	+11.52
7.	7.	7.	Microsoft Access	Relational DBMS	141.69	+1.15	-4.79
8.	8.	↑ 10.	Cassandra 🟡	Wide column store	107.31	+0.23	+29.22
9.	9.	↓ 8.	SQLite	Relational DBMS	101.71	+2.14	+8.73
10.	10.	↑ 13.	Redis	Key-value store	97.05	-2.16	+43.59
11.	11.	↓ 9.	SAP Adaptive Server	Relational DBMS	85.37	-0.97	+3.81
12.	12.	12.	Solr	Search engine	81.88	+0.40	+20.74
13.	13.	↓ 11.	Teradata	Relational DBMS	72.78	+3.33	+10.15
14.	14.	↑ 16.	HBase	Wide column store	60.73	+3.59	+25.59
15.	↑ 16.	↑ 19.	Elasticsearch	Search engine	58.92	+6.09	+32.75

Figura 6.15 Os 15 principais bancos de dados classificados por popularidade de acordo com DB-Engines.com em março de 2015

A Figura 6.15 mostra que, com 9 entradas, os bancos de dados relacionais ainda dominavam os 15 primeiros no momento em que este livro foi escrito e, com a chegada do NewSQL, ainda não podemos excluí-los. Neo4j, o banco de dados gráfico mais popular, pode ser encontrado na posição 23 no momento em que este artigo foi escrito, com Titan na posição 53.

Agora que você viu cada um dos tipos de banco de dados NoSQL , é hora de colocar a mão na massa com um deles.

6.2 Estudo de caso: Que doença é essa?

Já aconteceu com muitos de nós: você tem sintomas médicos repentinos e a primeira coisa que faz é pesquisar no Google qual doença os sintomas podem indicar; então você decide se vale a pena consultar um médico. Um mecanismo de busca na web é adequado para isso, mas um banco de dados mais dedicado seria melhor. Bancos de dados como este existem e são bastante avançados; eles podem ser quase uma versão virtual do Dr. House, um brilhante diagnosticador da série de TV *House MD*. Mas eles são baseados em dados bem protegidos e nem todos são acessíveis ao público. Além disso, embora as grandes empresas farmacêuticas e os hospitais avançados tenham acesso a estes médicos virtuais, muitos clínicos gerais ainda estão presos apenas aos seus livros. Esta assimetria de informações e recursos não é apenas triste e perigosa, ela nem precisa existir. Se um mecanismo de busca simples e específico para doenças fosse usado por todos os médicos de clínica geral do mundo, muitos erros médicos poderiam ser evitados.

Neste estudo de caso, você aprenderá como construir esse mecanismo de busca aqui, embora usando apenas uma fração dos dados médicos que estão disponíveis gratuitamente. Para resolver o problema, você usará um banco de dados NoSQL moderno chamado Elasticsearch para armazenar os dados, e o

processo de ciência de dados para trabalhar com os dados e transformá-los em um recurso rápido e fácil de pesquisar. Veja como você aplicará o processo:

- 1 *Definir o objetivo da pesquisa.*
- 2 *Coleta de dados* – Você obterá seus dados da Wikipedia. Existem mais fontes por aí, mas para fins de demonstração, um único servirá.
- 3 *Preparação de dados*—Os dados da Wikipédia podem não ser perfeitos em seu formato atual. Você aplicará algumas técnicas para mudar isso.
- 4 *Exploração de dados* — Seu caso de uso é especial porque a etapa 4 do processo de ciência de dados também é o resultado final desejado: você deseja que seus dados sejam fáceis de explorar.
- 5 *Modelagem de dados* – Nenhuma modelagem de dados real é aplicada neste capítulo. As matrizes de termos de documentos usadas para pesquisa costumam ser o ponto de partida para pesquisas avançadas. modelagem de tópicos. Não entraremos nisso aqui.
- 6 *Apresentando resultados*—Para tornar os dados pesquisáveis, você precisaria de uma interface de usuário como um site onde as pessoas podem consultar e recuperar informações sobre doenças. Nisso capítulo você não chegará ao ponto de construir uma interface real. Seu objetivo secundário: traçar o perfil de uma categoria de doença por meio de suas palavras-chave; você chegará a este estágio dos dados processo científico porque você o apresentará como uma nuvem de palavras, como a da Figura 6.16.

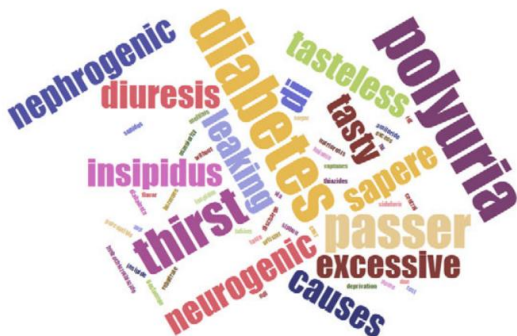


Figura 6.16 Um exemplo de nuvem de palavras em palavras-chave não ponderadas sobre diabetes

Para acompanhar o código, você precisará destes itens:

- Uma sessão Python com as bibliotecas `elasticsearch-py` e `Wikipedia` instaladas (pip instale o `elasticsearch` e o pip instale a `wikipedia`)
- Uma instância do `Elasticsearch` configurada localmente; consulte o apêndice A para obter instruções de instalação
- A biblioteca `IPython`

**NOTA** O código deste capítulo está disponível para download no Manning website para este livro em <https://manning.com/books/introduzindo-data-science> e está no formato `IPython`.



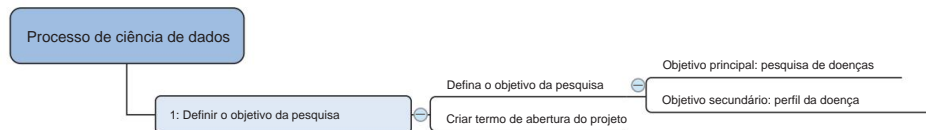
**Elasticsearch: o mecanismo de pesquisa de código aberto/banco de dados NoSQL** Para

resolver o problema em questão, diagnosticar uma doença, o banco de dados NoSQL que você usará é o Elasticsearch. Assim como o MongoDB, o Elasticsearch é um armazenamento de documentos. Mas, diferentemente do MongoDB, o Elasticsearch é um mecanismo de busca. Enquanto o MongoDB é excelente para realizar cálculos complexos e tarefas MapReduce, o objetivo principal do Elasticsearch é a pesquisa de texto completo. O Elasticsearch fará cálculos básicos em dados numéricos indexados, como soma, contagens, mediana, média, desvio padrão e assim por diante, mas em essência continua sendo um mecanismo de busca.

O Elasticsearch foi desenvolvido com base no Apache Lucene, o mecanismo de busca Apache criado em 1999. O Lucene é notoriamente difícil de manusear e é mais um alicerce para aplicativos mais fáceis de usar do que uma solução ponta a ponta em si. Mas o Lucene é um mecanismo de busca extremamente poderoso, e o Apache Solr o seguiu em 2004, abrindo para uso público em 2006. O Solr (uma plataforma de busca corporativa de código aberto) é construído sobre o Apache Lucene e neste momento ainda é o mais versátil e popular mecanismo de pesquisa de código aberto. Solr é uma ótima plataforma e vale a pena investigar se você se envolver em um projeto que requer um mecanismo de busca. Em 2010, o Elasticsearch surgiu, ganhando popularidade rapidamente. Embora o Solr ainda possa ser difícil de instalar e configurar, mesmo para projetos pequenos, o Elasticsearch não poderia ser mais fácil. O Solr ainda tem uma vantagem no número de plug-ins possíveis que expandem sua funcionalidade principal, mas o Elasticsearch está se atualizando rapidamente e hoje seus recursos são de qualidade comparável.

**6.2.1 Passo 1: Definir o objetivo da pesquisa**

Você consegue diagnosticar uma doença até o final deste capítulo, usando apenas seu próprio computador doméstico e o software e os dados gratuitos disponíveis? Saber o que você quer fazer e como fazer é o primeiro passo no processo de ciência de dados, conforme mostrado na figura 6.17.



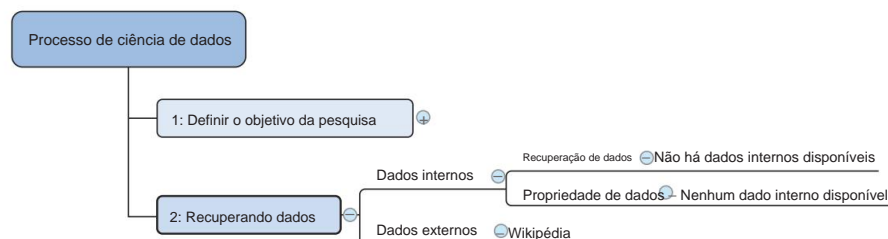
**Figura 6.17** Etapa 1 no processo de ciência de dados: definição do objetivo da pesquisa

- O seu objectivo principal é criar um motor de busca de doenças que ajude os médicos de clínica geral no diagnóstico de doenças.
- Seu objetivo secundário é traçar o perfil de uma doença: quais palavras-chave a distinguem outras doenças?

Este objetivo secundário é útil para fins educativos ou como contributo para utilizações mais avançadas, como a deteção de propagação de epidemias através do recurso às redes sociais. Com o seu objetivo de pesquisa e um plano de ação definido, vamos passar para a etapa de recuperação dos dados.

### 6.2.2 Etapas 2 e 3: Recuperação e preparação de dados

A recuperação e a preparação de dados são duas etapas distintas no processo de ciência de dados e, embora isso continue sendo verdade para o estudo de caso, exploraremos ambas na mesma seção. Dessa forma, você pode evitar a configuração de armazenamento intermediário local e preparar os dados imediatamente enquanto os dados estão sendo recuperados. Vejamos onde estamos no processo de ciência de dados (veja a figura 6.18).



**Figura 6.18** Etapa 2 do processo de ciência de dados: recuperação de dados. Neste caso não há dados internos; todos os dados serão obtidos da Wikipédia.

Conforme mostrado na figura 6.18 você tem duas fontes possíveis: dados internos e dados externos.

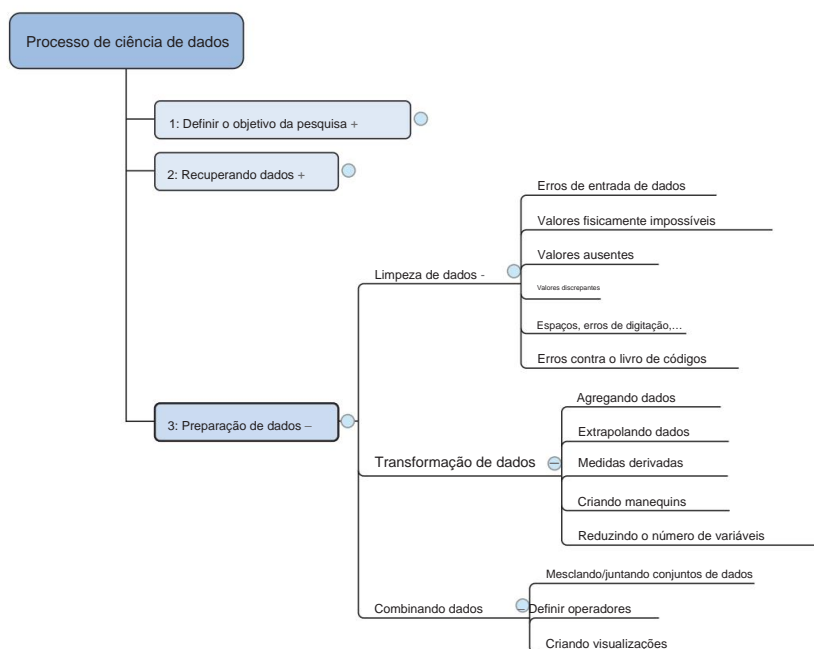
- *Dados internos* – você não tem informações sobre doenças por aí. Se você atualmente trabalha para uma empresa farmacêutica ou hospital, poderá ter mais sorte.
- *Dados externos* —Tudo o que você pode usar neste caso são dados externos. Você tem várias possibilidades, mas você irá com a Wikipédia.

Ao extrair os dados da Wikipédia, você precisará armazená-los em seu índice local do Elastic-search, mas antes de fazer isso, você precisará preparar os dados. Depois que os dados entram no índice do Elasticsearch, eles não podem ser alterados; tudo o que você pode fazer é consultá-lo.

Veja a visão geral da preparação de dados na figura 6.19.

Conforme mostrado na figura 6.19, há três categorias distintas de preparação de dados a serem consideradas:

- *Limpeza de dados* – Os dados que você extrai da Wikipédia podem estar incompletos ou errados. Erros de entrada de dados e de ortografia são possíveis – mesmo informações falsas não são excluídas. Felizmente, você não precisa que a lista de doenças seja exaustiva e pode lidar com erros ortográficos na hora da pesquisa; mais sobre isso mais tarde. Graças à biblioteca Python da Wikipédia, os dados textuais que você receberá já estão bastante limpos. Se você raspar manualmente, precisará adicionar limpeza de HTML, removendo todas as tags HTML. A verdade é que a pesquisa de texto completo tende a ser bastante robusta em relação a erros comuns, como valores incorretos. Mesmo que você tenha inserido tags HTML de propósito, é improvável que elas influenciem os resultados; as tags HTML são muito diferentes da linguagem normal para interferir.



**Figura 6.19** Etapa 3 do processo de ciência de dados: preparação de dados

• *Transformação de dados* – você não precisa transformar muito os dados neste momento; você deseja pesquisá-lo como está. Mas você fará a distinção entre título da página, nome da doença e corpo da página. Esta distinção é quase obrigatória para a interpretação dos resultados da pesquisa.

• *Combinação de dados* — Neste caso, todos os dados são extraídos de uma única fonte, portanto não há necessidade real de combinar dados. Uma possível extensão deste exercício seria obter dados sobre doenças de outra fonte e fazer a correspondência entre as doenças. Esta não é uma tarefa trivial porque nenhum identificador exclusivo está presente e os nomes geralmente são ligeiramente diferentes.

Você pode fazer a limpeza de dados em apenas dois estágios: ao usar o programa Python que conecta a Wikipédia ao Elasticsearch e ao executar o sistema de indexação interna do Elasticsearch: • *Python* — *aqui* você define quais dados permitirão que

sejam armazenados pelo seu armazenamento de documentos, mas você não limpará ou transformará os dados neste estágio, porque o Elasticsearch é melhor nisso com menos esforço.

• *Elasticsearch* — O Elasticsearch cuidará da manipulação de dados (criação do índice) nos bastidores. Você ainda pode influenciar esse processo, e fará isso de forma mais explícita posteriormente neste capítulo.

Agora que você tem uma visão geral das etapas a seguir, vamos trabalhar. Se você seguiu seguindo as instruções no apêndice, agora você deve ter uma instância local do Elastic-search instalada e em execução. Primeiro vem a recuperação de dados: você precisa de informações sobre as diferentes doenças. Você tem várias maneiras de obter esse tipo de dados. Você poderia perguntar às empresas para obter seus dados ou obter dados do Freebase ou de outras fontes de dados abertas e gratuitas. Adquirir seus dados pode ser um desafio, mas neste exemplo você os extrairá da Wikipédia. Isso é um pouco irônico porque as pesquisas no próprio site da Wikipedia são tratadas por Elasticsearch. A Wikipedia costumava ter seu próprio sistema construído em cima do Apache Lucene, mas tornou-se insustentável e, a partir de janeiro de 2014, a Wikipédia começou a usar Em vez disso, Elasticsearch.

A Wikipédia possui uma página Listas de doenças, conforme mostrado na figura 6.20. A partir daqui você pode pegue emprestado os dados das listas alfabéticas.

## Lists of diseases

From Wikipedia, the free encyclopedia  
(Redirected from [List of diseases](#))

A **medical condition** is a broad term that includes all diseases and disorders.

A **disease** is an abnormal condition affecting the body of an organism.

A **disorder** is a functional abnormality or disturbance.

- [List of cancer types](#)
- [List of cutaneous conditions](#)
- [List of endocrine diseases](#)



Figura 6.20 Página Listas de doenças da Wikipédia, o ponto de partida para sua recuperação de dados

Você sabe quais dados deseja; agora vá pegá-lo. Você pode baixar todo o despejo de dados da Wikipédia. Se desejar, você pode baixá-lo em [http://meta.wikimedia.org/wiki/Data\\_dump\\_torrents#enwiki](http://meta.wikimedia.org/wiki/Data_dump_torrents#enwiki).

Claro, se você indexasse toda a Wikipédia, o índice acabaria exigindo cerca de 40 GB de armazenamento. Sinta-se à vontade para usar esta solução, mas para preservar o armazenamento e a largura de banda, nos limitaremos neste livro a extrair apenas o dados que pretendemos usar. Outra opção é raspar as páginas necessárias. Como o Google, você pode fazer um programa rastrear as páginas e recuperar todo o arquivo renderizado HTML. Isso resolveria o problema, mas você acabaria com o HTML real, então precisaria para limpar isso antes de indexá-lo. Além disso, a menos que você seja o Google, os sites também não são gosta de rastreadores que raspam suas páginas da web. Isso cria uma quantidade desnecessariamente alta de tráfego, e se um número suficiente de pessoas enviar rastreadores, isso pode levar o servidor HTTP ao seu

joelhos, estragando a diversão de todos. Enviar bilhões de solicitações ao mesmo tempo é também é uma das maneiras pelas quais os ataques de negação de serviço (DoA) são executados. Se você precisar raspar um site, criar um script em um intervalo de tempo entre cada solicitação de página. Dessa forma, seu scraper imita mais de perto o comportamento de um visitante regular do site e você não explodir seus servidores.

Felizmente, os criadores da Wikipédia são inteligentes o suficiente para saber que isto é exatamente o que aconteceria com toda essa informação aberta a todos. Eles colocaram uma API em local de onde você pode extrair suas informações com segurança. Você pode ler mais sobre isso em [http://www.mediawiki.org/wiki/API:Main\\_page](http://www.mediawiki.org/wiki/API:Main_page).

Você usará a API. E Python não seria Python se já não tivesse um biblioteca para fazer o trabalho. Na verdade, existem vários, mas o mais fácil será suficiente para o seu necessidades: Wikipédia.

Ative seu ambiente virtual Python e instale todas as bibliotecas necessárias para o resto do livro:

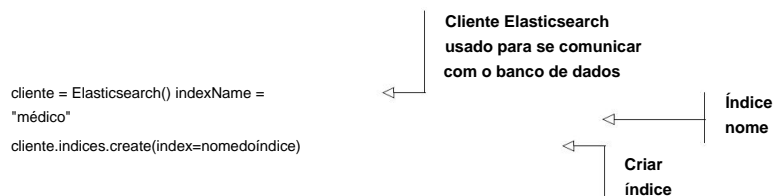
```
pip instalar wikipedia
pip instalar Elasticsearch
```

Você usará a Wikipédia para acessar a Wikipédia. Elasticsearch é o principal Elasticsearch Biblioteca Python; com ele você pode se comunicar com seu banco de dados.

Abra seu interpretador Python favorito e importe as bibliotecas necessárias:

```
da importação do elasticsearch Elasticsearch
importar Wikipédia
```

Você extrairá dados da API da Wikipédia e ao mesmo tempo indexará em seu instância local do Elasticsearch, então primeiro você precisa prepará-la para aceitação de dados.



A primeira coisa que você precisa é de um cliente. `Elasticsearch()` pode ser inicializado com um endereço, mas o padrão é `localhost:9200`. `Elasticsearch()` e `Elasticsearch('localhost:9200')` são, portanto, a mesma coisa: seu cliente está conectado ao seu local Nó Elasticsearch. Então você cria um índice chamado "medical". Se tudo correr bem, você deverá ver uma resposta "reconhecido:true", conforme mostrado na figura 6.21.

O Elasticsearch afirma não ter esquema, o que significa que você pode usar o Elasticsearch sem definir um esquema de banco de dados e sem informar ao Elasticsearch que tipo de dados ele

```
In [7]: client = Elasticsearch() #elasticsearch client used to communicate with the database
        indexName = "medical" #the index name
        #client.indices.delete(index=indexName) #delete an index
        client.indices.create(index=indexName) #create an index

Out[7]: {u'acknowledged': True}
```

**Figura 6.21 Criando um índice Elasticsearch com Python-Elasticsearch**

precisa esperar. Embora isso seja verdade para casos simples, você não pode evitar ter um esquema no longo prazo, então vamos criar um, conforme mostrado na listagem a seguir.

#### Listagem 6.1 Adicionando um mapeamento ao tipo de documento

```
DiseaseMapping =
    { 'propriedades': {
        'nome': {'tipo': 'string'},
        'título': {'tipo': 'string'},
        'texto completo': {'tipo': 'string'}
    }
}

client.indices.put_mapping(index=nomedoíndice,
doc_type='doenças',body=diseaseMapping )
```

Definir um mapeamento e atribuí-lo ao tipo de documento da doença.

O tipo de documento "doenças" é atualizado com um mapeamento. Agora definimos os dados que ele deve esperar.

Dessa forma você informa ao Elasticsearch que seu índice terá um tipo de documento chamado "doença" e você fornece o tipo de campo para cada um dos campos. Você tem três campos em um documento de doença: nome, título e texto completo, todos do tipo string. Se você não tivesse fornecido o mapeamento, o Elasticsearch teria adivinhado seus tipos olhando para a primeira entrada que recebeu. Se não reconhecesse o campo como booleano, double, float, long, inteiro ou data, seria definido como string. Neste caso, você não foi necessário especificar manualmente o mapeamento.

Agora vamos passar para a Wikipédia. A primeira coisa que você deseja fazer é buscar a lista de página de doenças, porque este é o seu ponto de entrada para uma exploração mais aprofundada:

```
dl = wikipedia.page("Listas_de_doenças")
```

Agora você tem sua primeira página, mas está mais interessado nas páginas de listagem porque eles contêm ligações com as doenças. Confira os links:

```
dl.links
```

A página Lista de doenças vem com mais links do que você usará. A Figura 6.22 mostra o listas alfabéticas começando no décimo sexto link.

```
dl = wikipedia.page("Listas_de_doenças")
dl.links
```

```
In [9]: dl = wikipedia.page("Lists_of_diseases")
        dl.links

Out[9]: [u'Airborne disease',
        u'Contagious disease',
        u'Cryptogenic disease',
        u'Disease',
        u'Disseminated disease',
        u'Endocrine disease',
        u'Environmental disease',
        u'Eye disease',
        u'Lifestyle disease',
        u'List of abbreviations for diseases and disorders',
        u'List of autism-related topics',
        u'List of basic exercise topics',
        u'List of cancer types',
        u'List of communication disorders',
        u'List of cutaneous conditions',
        u'List of diseases (0\u2013139)',
        u'List of diseases (A)',
        u'List of diseases (B)']
```

**Figura 6.22** Links na página da Wikipédia Listas de doenças. Tem mais links do que você precisa.

Esta página possui uma variedade considerável de links, mas apenas as listas alfabéticas lhe interessam, portanto, mantenha apenas estas:

DiseaseListArray = [] para link em  
dl.links[15:42]: tente:

```
DiseaseListArray.append(wikipedia.page(link))
exceto exceção, e: print str(e)
```

Você provavelmente notou que o subconjunto é codificado, porque você sabe que eles são da 16ª à 43ª entradas na matriz. Se a Wikipedia adicionasse um único link antes daqueles nos quais você está interessado, isso prejudicaria os resultados. Uma prática melhor seria usar expressões regulares para esta tarefa. Para fins de exploração, codificar os números de entrada é adequado, mas se as expressões regulares forem uma segunda natureza para você ou se você pretende transformar esse código em um trabalho em lote, as expressões regulares são recomendadas. Você pode encontrar mais informações sobre eles em <https://docs.python.org/2/howto/regex.html>.

Uma possibilidade para uma versão regex seria o seguinte trecho de código.

DiseaseListArray = [] check =  
re.compile("Lista de doenças") para link em dl.links:

se check.match(link): tente:

```
doençaListArray.append(wikipedia.page(link))
exceto exceção, e: print str(e)
```

```
In [16]: diseaseListArray

Out[16]: [<WikipediaPage 'List of diseases (0-9)'>,
<WikipediaPage 'List of diseases (A)'>,
<WikipediaPage 'List of diseases (B)'>,
<WikipediaPage 'List of diseases (C)'>,
<WikipediaPage 'List of diseases (D)'>,
<WikipediaPage 'List of diseases (E)'>,
<WikipediaPage 'List of diseases (F)'>,
<WikipediaPage 'List of diseases (G)'>,
<WikipediaPage 'List of diseases (H)'>]
```

Figura 6.23 Primeira lista de doenças da Wikipedia, “lista de doenças (0-9)”

A Figura 6.23 mostra as primeiras entradas do que você procura: as próprias doenças.

doençaListArray[0].links

É hora de indexar as doenças. Uma vez indexados, tanto a entrada quanto a preparação dos dados estão efetivamente concluídas, conforme mostrado na listagem a seguir.

Listagem 6.2 Indexação de doenças da Wikipédia

Looping através de doença listas.

```
lista de verificação = [[ "0","1","2","3","4","5","6","7","8","9"],
["A"],["B"],["C"],["D"],["E"],["F"],["G"],["H"],
["I"],["J"],["K"],["L"],["M"],["N"],["O"],["P"],
["Q"],["R"],["S"],["T"],["U"],["V"],["W"],["X"],[" Y"],["Z"]] docType = 'doenças'

para lista de doençasNumber, lista de doenças em enumerar(diseaseListArray):
    para doenças em DiseaseList.links:
        tentar:
            se doença[0] em checkList[diseaseListNumber]
e doença[0:3] != "Lista":
                páginaatual = wikipedia.page(doença)
                cliente.index(index=nomedoíndice,
doc_type=docType,id = doença, corpo={"nome": doença,
"title":currentPage.title, "fulltext":currentPage.content})

                exceto exceção, e: print str(e)
```

A lista de verificação é uma matriz contendo uma matriz de primeiros caracteres permitidos. Se uma doença não obedecer, ignore-a.

Documento digite você indexará.

Percorrendo listas de links para cada lista de doenças.

Primeiro verifique se é uma doença e depois indexe.

Como cada uma das páginas da lista terá links desnecessários, verifique se há uma entrada uma doença. Você indica para cada lista com qual personagem a doença começa, então você verifica por esta. Além disso, você exclui os links que começam com “lista” porque eles aparecerão quando você chegar à lista L de doenças. A verificação é um tanto ingênua, mas o custo de ter algumas entradas indesejadas é bastante baixo porque os algoritmos de busca excluirão resultados irrelevantes quando você começa a consultar. Para cada doença você indexa a doença nome e o texto completo da página. O nome também é usado como ID de índice; isso é útil



para vários recursos avançados do Elasticsearch, mas também para pesquisa rápida no navegador. Por exemplo, tente este URL no seu navegador: `http://localhost:9200/medical/diseases/11%20beta%20hidroxilase%20deficiência`. O título é indexado separadamente; na maioria dos casos, o nome do link e o título da página serão idênticos e às vezes o título será contê-lo um nome alternativo para a doença.

Com pelo menos algumas doenças indexadas é possível fazer uso do Elasticsearch URI para pesquisas simples. Dê uma olhada em uma busca de corpo inteiro pela palavra *dor de cabeça* na Figura 6.24. Você já pode fazer isso durante a indexação; Elasticsearch pode atualizar um índice e retornar consultas sobre ele ao mesmo tempo.

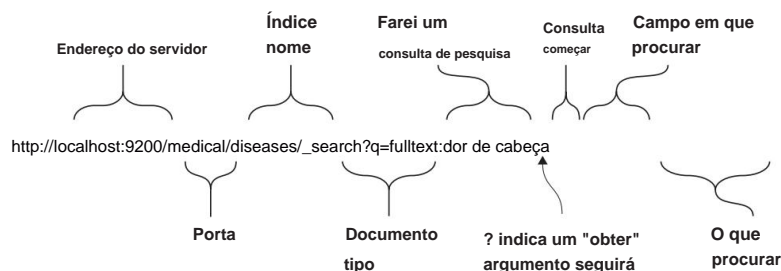


Figura 6.24 O exemplo de construção de URL do Elasticsearch

Se você não consultar o índice, ainda poderá obter um poucos resultados sem saber nada sobre o índice. Especificando `http://localhost:9200/medical/diseases/_search` retornará o primeiro cinco resultados. Para uma visão mais estruturada sobre o dados você pode solicitar o mapeamento deste tipo de documento em `http://localhost:9200/medical/doenças/_mapping?pretty`. O argumento `pretty` get mostra o JSON retornado em um formato mais legível, como pode ser visto na figura 6.25. O mapeamento parece ser da maneira que você especificou: todos os campos são do tipo string.

A URL do Elasticsearch é certamente útil, mas não será suficiente para suas necessidades. Você ainda tem doenças para diagnosticar, e para isso enviará POST solicitações para o Elasticsearch por meio do seu Elasticsearch Biblioteca Python.

Com a recuperação e preparação de dados concluídas, você pode prosseguir para a exploração de seus dados.

```
{
  "medical" : {
    "mappings" : {
      "diseases" : {
        "properties" : {
          "fulltext" : {
            "type" : "string"
          },
          "name" : {
            "type" : "string"
          },
          "title" : {
            "type" : "string"
          }
        }
      }
    }
  }
}
```

Figura 6.25 Mapeamento de tipo de documento de doenças via URL do Elasticsearch

### 6.2.3 Etapa 4: Exploração de dados

Não é lúpus. Nunca é lúpus!

—Dr. Casa da Casa MD

A exploração de dados é o que marca este estudo de caso, porque o objetivo principal do projeto (diagnóstico de doenças) é uma forma específica de explorar os dados através da consulta de sintomas de doenças. A Figura 6.26 mostra diversas técnicas de exploração de dados, mas neste caso não é gráfica: interpretação dos resultados da consulta de pesquisa de texto.

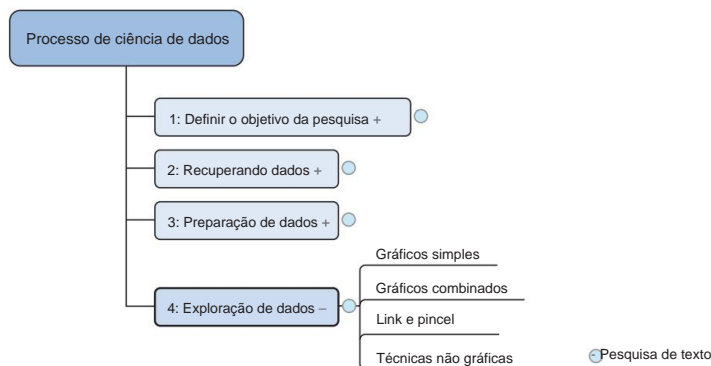


Figura 6.26 Etapa 4 do processo de ciência de dados: exploração de dados

O momento da verdade chegou: você pode encontrar certas doenças alimentando seu mecanismo de busca com seus sintomas? Vamos primeiro ter certeza de que você tem o básico instalado e funcionando. Importe a biblioteca Elasticsearch e defina as configurações de pesquisa global:

```
from elasticsearch import Cliente Elasticsearch = Elasticsearch() indexName = "medical"
```

```
docType="doenças" searchFrom = 0
```

```
tamanho da pesquisa = 3
```

Você retornará apenas os três primeiros resultados; o padrão é cinco.

Elasticsearch possui uma linguagem de consulta JSON elaborada ; toda pesquisa é uma solicitação POST para o servidor e será respondida com uma resposta JSON . Aproximadamente, a linguagem consiste em três grandes partes: consultas, filtros e agregações. Uma *consulta* pega palavras-chave de pesquisa e as passa por um ou mais analisadores antes que as palavras sejam pesquisadas no índice. Iremos nos aprofundar nos analisadores um pouco mais adiante neste capítulo. Um *filtro* usa palavras-chave como uma consulta, mas não tenta analisar o que você fornece; ele filtra as condições que oferecemos. Os filtros são, portanto, menos complexos, mas muitas vezes mais eficientes porque

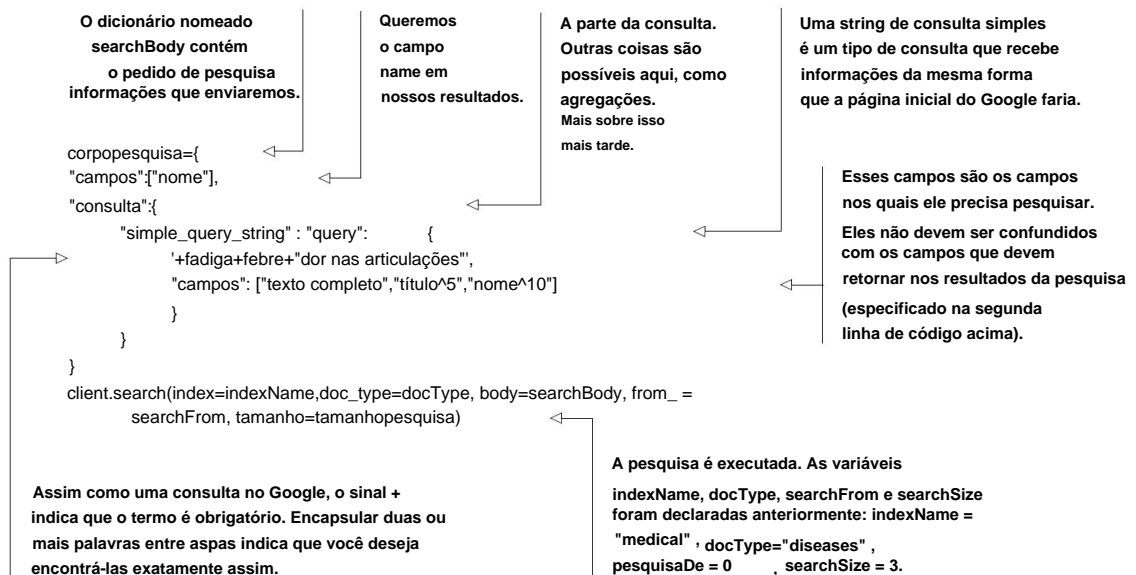
eles também são armazenados temporariamente no Elasticsearch caso você use o mesmo filtro duas vezes. As *agregações* podem ser comparadas ao grupo SQL ; grupos de palavras serão criados e, para cada grupo, estatísticas relevantes poderão ser calculadas. Cada um desses três compartimentos possui inúmeras opções e recursos, facilitando a elaboração de todo o linguagem aqui impossível. Felizmente, não há necessidade de entrar na complexidade que as consultas do Elasticsearch podem representar. Usaremos a "linguagem de consulta de string de consulta", uma forma de consultar os dados que se assemelham muito à linguagem de consulta de pesquisa do Google. Se, por exemplo, você deseja que um termo de pesquisa seja obrigatório, adicione um sinal de mais (+); se você quiser para excluir o termo de pesquisa, use um sinal de menos (-). Consultar o Elasticsearch não é recomendado porque diminui o desempenho; o mecanismo de pesquisa primeiro precisa traduzir a string de consulta para sua linguagem de consulta JSON nativa . Mas para seus propósitos, será funciona bem; além disso, o desempenho não será um fator importante nos vários milhares de registros que você tem em seu índice. Agora é hora de consultar os dados da sua doença.

#### OBJETIVO PRIMÁRIO DO PROJETO: DIAGNÓSTICO DE UMA DOENÇA PELOS SEUS SINTOMAS

Se você já viu a popular série de televisão *House MD*, a frase "Nunca é lúpus" pode parecer familiar. O lúpus é um tipo de doença autoimune, em que o sistema imunológico do corpo sistema ataca partes saudáveis do corpo. Vamos ver quais sintomas seu mecanismo de pesquisa precisaria determinar se você está procurando por lúpus.

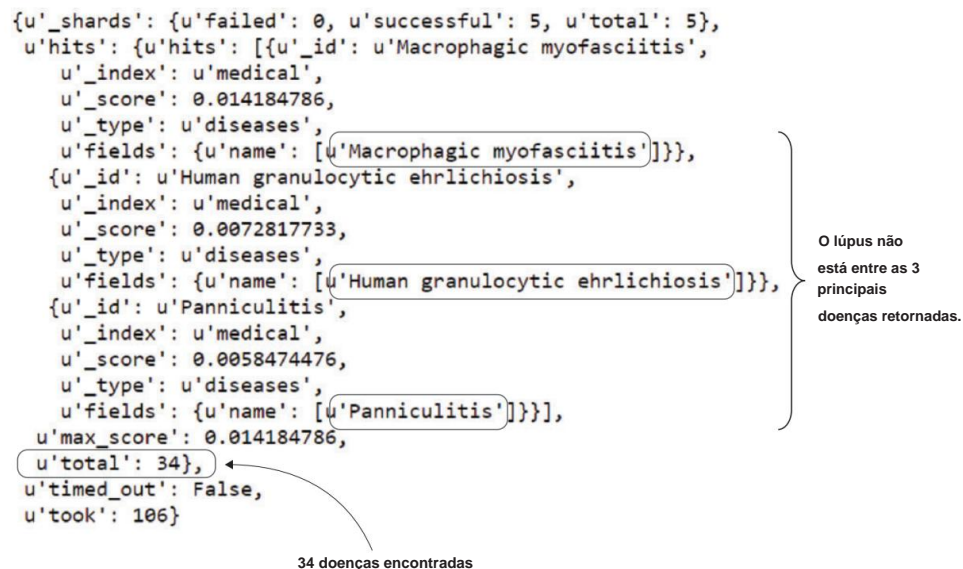
Comece com três sintomas: fadiga, febre e dores nas articulações. Seu paciente imaginário tem todos os três (e mais), então torne-os obrigatórios adicionando um sinal de mais antes de cada um:

#### Listagem 6.3 Consulta Elasticsearch de "string de consulta simples" com três palavras-chave obrigatórias



No `searchBody`, que possui uma estrutura JSON, você especifica os campos que gostaria que fossem retornados, neste caso o nome da doença deve ser suficiente. Você usa a sintaxe da string de consulta para pesquisar em todos os campos indexados: texto completo, título e nome. Ao adicionar pode-se dar um <sup>^</sup> você peso a cada campo. Se um sintoma ocorre no título, é cinco vezes mais importante que no texto aberto; se ocorrer no próprio nome, é considerado dez vezes mais importante. Observe como “dor nas articulações” está entre aspas. Se você não tivesse os sinais "", *articulação* e *dor* seriam consideradas duas palavras-chave separadas, em vez de uma única frase. No Elasticsearch isso é chamado de *correspondência de frase*.

Vejamos os resultados na figura 6.27.



```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
  u'hits': {u'hits': [{u'_id': u'Macrophagic myofasciitis',
    u'_index': u'medical',
    u'_score': 0.014184786,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Macrophagic myofasciitis']}},
    {u'_id': u'Human granulocytic ehrlichiosis',
    u'_index': u'medical',
    u'_score': 0.0072817733,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Human granulocytic ehrlichiosis']}},
    {u'_id': u'Panniculitis',
    u'_index': u'medical',
    u'_score': 0.0058474476,
    u'_type': u'diseases',
    u'fields': {u'name': [u'Panniculitis']}},
    u'max_score': 0.014184786,
    u'total': 34},
  u'timed_out': False,
  u'took': 106}
```

O lúpus não está entre as 3 principais doenças retornadas.

34 doenças encontradas

Figura 6.27 Primeira pesquisa de Lupus com 34 resultados

A Figura 6.27 mostra os três principais resultados de 34 doenças correspondentes. Os resultados são classificados pela pontuação correspondente, a variável `_score`. A pontuação correspondente não é algo simples de explicar; leva em consideração o quão bem a doença corresponde à sua consulta e quantas vezes uma palavra-chave foi encontrada, os pesos que você deu e assim por diante.

Atualmente, o lúpus nem aparece nos três primeiros resultados. Felizmente para você, o lúpus tem outro sintoma distinto: erupção na pele. A erupção cutânea nem sempre aparece no rosto da pessoa, mas acontece e é daí que o lúpus ganhou esse nome: a erupção facial faz com que as pessoas se pareçam vagamente com um lobo. Seu paciente tem uma erupção cutânea, mas não a erupção característica no rosto, então adicione “erupção cutânea” aos sintomas sem mencionar o rosto.

"consulta": '+fadiga+febre+"dor nas articulações"+erupção cutânea',

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
 u'_hits': {u'_hits': [{u'_id': u'Human granulocytic ehrlichiosis',
    u'_index': u'medical',
    u'_score': 0.009902062,
    u'_type': u'diseases',
    u'_fields': {u'name': [u'Human granulocytic ehrlichiosis']}},
 {u'_id': u'Lupus erythematosus',
    u'_index': u'medical',
    u'_score': 0.009000875,
    u'_type': u'diseases',
    u'_fields': {u'name': [u'Lupus erythematosus']}},
 {u'_id': u'Panniculitis',
    u'_index': u'medical',
    u'_score': 0.007950994,
    u'_type': u'diseases',
    u'_fields': {u'name': [u'Panniculitis']}},
 u'_max_score': 0.009902062,
 u'_total': 6},
 u'_timed_out': False,
 u'_took': 15}
```

**Figura 6.28** Lúpus segunda tentativa de pesquisa com seis resultados e lúpus entre os três primeiros

Os resultados da nova pesquisa são mostrados na figura 6.28.

Agora os resultados foram reduzidos a seis e o lúpus está entre os três primeiros. Neste ponto, o mecanismo de busca diz que a *Erliquiose Granulocítica Humana* (HGE) é mais provável. HGE é uma doença transmitida por carrapatos, como a infame doença de Lyme. A esta altura, um médico capaz já teria descoberto qual doença assola seu paciente, porque na determinação de doenças muitos fatores estão em jogo, mais do que você pode inserir em seu humilde mecanismo de busca. Por exemplo, a erupção cutânea ocorre apenas em 10% dos pacientes com HGE e em 50% dos pacientes com lúpus. O lúpus surge lentamente, enquanto o HGE é desencadeado por uma picada de carrapato. Bancos de dados avançados de aprendizado de máquina alimentados com todas essas informações de forma mais estruturada poderiam fazer um diagnóstico com muito maior certeza. Dado que você precisa se contentar com as páginas da Wikipedia, você precisa de outro sintoma para confirmar que é lúpus. O paciente sente dor no peito, então adicione isso à lista.

```
"consulta": "+fadiga+febre+"dor nas articulações"+erupção cutânea+"dor no peito",
```

O resultado é mostrado na figura 6.29.

Parece que é lúpus. Demorou um pouco para chegar a essa conclusão, mas você chegou lá. É claro que você foi limitado na maneira como apresentou os sintomas ao Elasticsearch. Você usou apenas termos isolados (“fadiga”) ou frases literais (“dor nas articulações”). Isso funcionou neste exemplo, mas o Elasticsearch é mais flexível que isso. Ele pode pegar expressões regulares e fazer uma pesquisa difusa, mas isso está além do escopo deste livro, embora alguns exemplos estejam incluídos no código para download.

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
  u'hits': {u'hits': [{u'_id': u'Lupus erythematosus',
    u'_index': u'medical',
    u'_score': 0.010452312,
    u'_type': u'diseases',
    u'_fields': {u'name': [u'Lupus erythematosus']}}]},
  u'max_score': 0.010452312,
  u'total': 1},
  u'timed_out': False,
  u'took': 11}
```

**Figura 6.29** Lúpus terceira pesquisa: com sintomas suficientes para determinar deve ser lúpus

### TRATANDO ERROS DE ORTOGRAFIA: DAMERAU-LEVENSHTEIN

Digamos que alguém digitou “lupsu” em vez de “lupus”. Erros ortográficos acontecem o tempo todo e em todos os tipos de documentos criados pelo homem. Para lidar com isso, os cientistas de dados frequentemente use Damerau-Levenshtein. A distância Damerau-Levenshtein entre duas cordas é o número de operações necessárias para transformar uma string em outra. Quatro operações podem calcular a distância:

• *Exclusão* — *Exclui* um caractere da string.

• *Inserção* — *Adicione* um caractere à string.

• *Substituição* — *Substitua* um caractere por outro. Sem a substituição

contado como uma operação, mudar um caractere para outro levaria dois operações: uma exclusão e uma inserção.

• *Transposição de dois caracteres adjacentes* — *Troque* dois caracteres adjacentes.

Esta última operação (transposição) é o que faz a diferença entre o tradicional Distância Levenshtein e distância Damerau-Levenshtein. É esta última operação isso faz com que nosso erro ortográfico disléxico caia dentro de limites aceitáveis. Damerau-Levenshtein perdoa esses erros de transposição, o que o torna ótimo para pesquisa motores, mas também é usado para outras coisas, como calcular as diferenças entre Cadeias de DNA .

A Figura 6.30 mostra como é realizada a transformação de “lupsu” para “lupus” com uma única transposição.

Lobo → Lobo → Lúpus

**Figura 6.30** A transposição de caracteres adjacentes é uma das operações na distância Damerau-Levenshtein. Os outros três são inserção, exclusão e substituição.

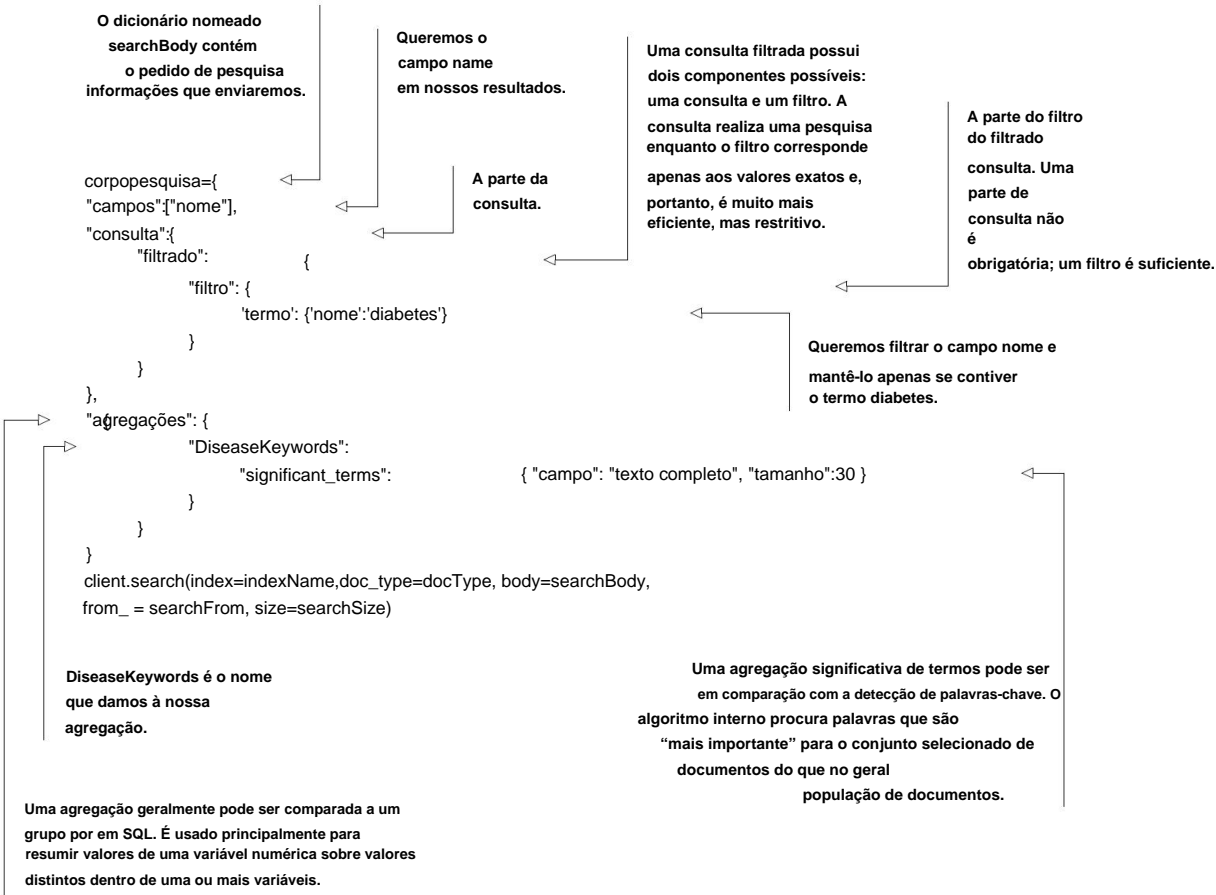
Com apenas isso você alcançou seu primeiro objetivo: *diagnosticar uma doença*. Mas não nos esqueçamos do objetivo secundário do projeto: *o perfil da doença*.

### OBJETIVO SECUNDÁRIO DO PROJETO: PERFIL DA DOENÇA

O que você deseja é uma lista de palavras-chave adequadas à doença selecionada. Para isso você usará a agregação de termos significativos. O cálculo da pontuação para determinar quais palavras são significativas é mais uma vez uma combinação de fatores, mas se resume aproximadamente a uma comparação.

do número de vezes que um termo é encontrado no conjunto de resultados em oposição a todos os outros documentos. Dessa forma, o Elasticsearch traça o perfil do seu conjunto de resultados, fornecendo as palavras-chave que o distingue dos outros dados. Vamos fazer isso com relação ao diabetes, uma doença comum que pode assumir muitas formas:

Listagem 6.4 Termos significativos Consulta do Elasticsearch para "diabetes"



Você vê o novo código aqui. Você se livrou da pesquisa de string de consulta e usou um filtro. O filtro é encapsulado na parte da consulta porque as consultas de pesquisa podem ser combinadas com filtros. Isso não ocorre neste exemplo, mas quando isso acontece, o Elastic-search aplicará primeiro o filtro muito mais eficiente antes de tentar a pesquisa. Se você Se você deseja pesquisar em um subconjunto de seus dados, é sempre uma boa ideia adicionar um filtro para primeiro criar este subconjunto. Para demonstrar isso, considere os dois trechos a seguir de código. Eles produzem os mesmos resultados, mas não são exatamente a mesma coisa.

Uma string de consulta simples procurando por “diabetes” no nome da doença:

```
"consulta":
  { "simple_query_string":
    { "consulta": 'diabetes', "campos":
      ["nome"] }
  }
```

Um termo filtro que filtra todas as doenças com “diabetes” no nome:

```
"consulta":{
  "filtrado":
    { "filtro": { 'termo':
      { 'nome':'diabetes' }
    }
}
```

Embora não apareça na pequena quantidade de dados à sua disposição, o filtro é muito mais rápido que a pesquisa. Uma consulta de pesquisa calculará uma pontuação de pesquisa para cada uma das doenças e as classificará de acordo, enquanto um filtro simplesmente filtra todas aquelas que não atendem. Um filtro é, portanto, muito menos complexo do que uma pesquisa real: pode ser “sim” ou “não” e isso fica evidente no resultado. A pontuação é 1 para tudo; nenhuma distinção é feita dentro do conjunto de resultados. A saída agora consiste em duas partes devido à agregação significativa de termos. Antes você só tinha hits; agora você tem acessos e agregações.

Primeiro, dê uma olhada nos acertos na figura 6.31.

Isso já deve parecer familiar, com uma exceção notável: todos os resultados têm pontuação 1. Além de ser mais fácil de executar, um filtro é armazenado em cache pelo Elasticsearch para

```
u'hits': {u'hits': [{u'_id': u'Diabetes mellitus',
  u'_index': u'medical',
  u'_score': 1.0,
  u'_type': u'diseases',
  u'fields': {u'name': [u'Diabetes mellitus']}},
{u'_id': u'Diabetes insipidus, nephrogenic type 3',
  u'_index': u'medical',
  u'_score': 1.0,
  u'_type': u'diseases',
  u'fields': {u'name': [u'Diabetes insipidus, nephrogenic type 3']}},
{u'_id': u'Ectodermal dysplasia arthrogryposis diabetes mellitus',
  u'_index': u'medical',
  u'_score': 1.0,
  u'_type': u'diseases',
  u'fields': {u'name': [u'Ectodermal dysplasia arthrogryposis diabetes mellitus']}},
u'max_score': 1.0,
u'total': 27},
u'timed_out': False,
u'took': 44}
```

Figura 6.31 Resultados de resultados da consulta filtrada com o filtro “diabetes” no nome da doença



um tempo. Dessa forma, as solicitações subsequentes com o mesmo filtro são ainda mais rápidas, resultando em uma enorme vantagem de desempenho em relação às consultas de pesquisa.

Quando você deve usar filtros e quando pesquisar consultas? A regra é simples: use filtros sempre que possível e use consultas de pesquisa para pesquisa de texto completo quando for necessária uma classificação entre os resultados para obter os resultados mais interessantes no topo.

Agora dê uma olhada nos termos significativos na figura 6.32.

```
{u'_shards': {u'failed': 0, u'successful': 5, u'total': 5},
  u'aggregations': {u'DiseaseKeywords': {u'buckets': [{u'bg_count': 18,
    u'doc_count': 9,
    u'key': u'siphon',
    u'score': 62.84567901234568},
    {u'bg_count': 18,
    u'doc_count': 9,
    u'key': u'diabainein',
    u'score': 62.84567901234568},
    {u'bg_count': 18,
    u'doc_count': 9,
    u'key': u'bainein',
    u'score': 62.84567901234568},
    {u'bg_count': 20,
    u'doc_count': 9,
    u'key': u'passer',
    u'score': 56.52777777777778},
    {u'bg_count': 14,
    u'doc_count': 7,
    u'key': u'ndi',
    u'score': 48.87997256515774},
```

**Figura 6.32** Agregação de termos significativos para diabetes, primeiras cinco palavras-chave

Se você observar as cinco primeiras palavras-chave da Figura 6.32, verá que as quatro principais estão relacionadas à origem do diabetes. O seguinte parágrafo da Wikipedia oferece ajuda:

A palavra diabetes (/ˈdaɪ.ˌɪbi.əti.ɪz/ ou /ˈdaɪ.ˌɪbi.əti.əs/) vem do latim diabe— te— s, que por sua vez vem do grego antigo ὑὑὑὑὑὑὑὑ (diabe— te— s) que literalmente significa “um transeunte; um sifão” [69]. O antigo médico grego Areteu da Capadócia (fl. Século I dC) usou essa palavra, com o significado pretendido de “descarga excessiva de urina”, como o nome da doença [70, 71, 72]. Em última análise, a palavra vem do grego ὑὑὑὑὑὑὑὑ ὑὑὑὑ (diabainein), que significa “passar através”, [69] que é composto de ὑὑὑ- (dia-), que significa “através” e ὑὑὑὑ ὑὑὑὑ (bainein), que significa “ir” [70].

A palavra “diabetes” foi registrada pela primeira vez em inglês, na forma diabetes, em um texto médico escrito por volta de 1425.

—Página da Wikipédia Diabetes\_mellitus

Isto lhe diz de onde vem a palavra *diabetes*: “um transeunte; um sifão” em grego. Também menciona *diabaineína* e *baineína*. Você deve saber que o mais

palavras-chave relevantes para uma doença seriam a definição e a origem reais. Felizmente nós pedimos 30 palavras-chave, então vamos escolher algumas mais interessantes, como *ndi*. *ndi* é um versão minúscula de *NDI*, ou "Diabetes Insipidus Nefrogênico", o mais comum forma adquirida de diabetes. Palavras-chave em minúsculas são retornadas porque é assim eles são armazenados no índice quando o passamos pelo analisador padrão durante a indexação. Não especificamos nada durante a indexação, então o analisador padrão foi usado por padrão. Outras palavras-chave interessantes entre as 30 primeiras são *avp*, um gene relacionado ao diabetes; *sede*, sintoma de diabetes; e *Amilorida*, um medicamento para diabetes. Essas palavras-chave parecem traçar o perfil do diabetes, mas faltam palavras-chave com vários termos; nós armazenamos apenas termos individuais no índice porque esse era o comportamento padrão. Certas palavras nunca aparecerão por si só porque não são usados com frequência, mas ainda são significativos quando usados em combinação com outros termos. Atualmente perdemos a relação entre certos termos. Veja o *avp*, por exemplo; se *avp* fosse sempre escrito em seu formulário completo "Diabetes Insipidus Nefrogênico", ele não seria atendido. Armazenando *n-gramas* (combinações de *n* número de palavras) ocupam espaço de armazenamento e usá-las para consultas ou agregações sobrecarrega o servidor de pesquisa. Decidir onde parar é um exercício de equilíbrio e depende dos seus dados e caso de uso.

Geralmente, os bigramas (combinação de dois termos) são úteis porque são significativos. bigramas existem na linguagem natural, embora 10 gramas nem tanto. Os conceitos-chave do bigram seriam úteis para o perfil de doenças, mas para criar esses termos significativos do bigrama agregações, você precisa delas armazenadas como bigramas em seu índice. Como muitas vezes acontece em ciência de dados, você precisará voltar várias etapas para fazer algumas alterações. Vamos voltar para a fase de preparação de dados.

6.2.4 Passo 3 revisitado: Preparação de dados para perfilamento de doenças

Não deveria ser surpresa que você tenha voltado à preparação de dados, como mostra a Figura 6.33. Afinal, o processo de ciência de dados é iterativo. Quando você indexou seus dados, você praticamente não fez nenhuma limpeza ou transformação de dados. Você pode adicionar limpeza de dados agora, por exemplo, interrompendo a filtragem de palavras. *Palavras irrelevantes* são palavras que são tão comuns que muitas vezes são descartados porque podem poluir os resultados. Nós não entrarei na filtragem de palavras interrompidas (ou outra limpeza de dados) aqui, mas sintá-se à vontade para tentar por você mesmo.

Para indexar bigramas você precisa criar seu próprio filtro de token e analisador de texto. A *filtro de token* é capaz de colocar transformações em tokens. Seu filtro de token específico

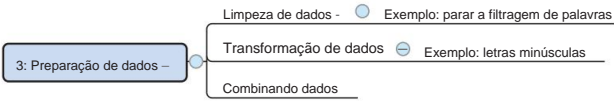


Figura 6.33 Etapa 3 do processo de ciência de dados: preparação de dados. A limpeza de dados para texto pode interromper a filtragem de palavras; a transformação de dados pode ser letras minúsculas.

precisa combinar tokens para criar n-gramas, também chamados *de telhas*. O tokenizador de pesquisa elástica padrão é chamado de tokenizer padrão e procurará limites de palavras, como o espaço entre as palavras, para cortar o texto em diferentes símbolos ou termos. Dê uma observe as novas configurações do seu índice de doença, conforme mostrado na listagem a seguir.

#### Listagem 6.5 Atualizando configurações de índice do Elasticsearch

```
configurações={
  "análise": {
    "filtro": {
      "meu_shingle_filter": {
        "type": "shingle",
        "min_shingle_size": 2,
        "max_shingle_size": 2,
        "output_unigrams": Falso
      }
    },
    "analisador": {
      "meu_shingle_analyzer": {
        "tipo": "personalizado",
        "tokenizador": "padrão",
        "filtro": [
          "minúsculas",
          "meu_shingle_filtro"
        ]
      }
    }
  }
}

cliente.indices.close(index=nomedoíndice)
client.indices.put_settings(index=indexName, corpo = configurações)
cliente.indices.open(index=nomedoíndice)
```

Antes de poder alterar determinadas configurações, o índice precisa ser fechado. Depois de alterar as configurações, você pode reabrir o índice.

Você cria dois novos elementos: o filtro de token chamado "meu filtro shingle" e um novo analisador chamado "my\_shingle\_analyzer". Como os n-gramas são tão comuns, o Elastic-search vem com um tipo de filtro de token shingle integrado. Tudo que você precisa dizer é que você deseja os bigramas "min\_shingle\_size": 2, "max\_shingle\_size": 2, conforme mostrado na figura 6.34. Você poderia optar por trigramas e superiores, mas para fins de demonstração isso será suficiente.

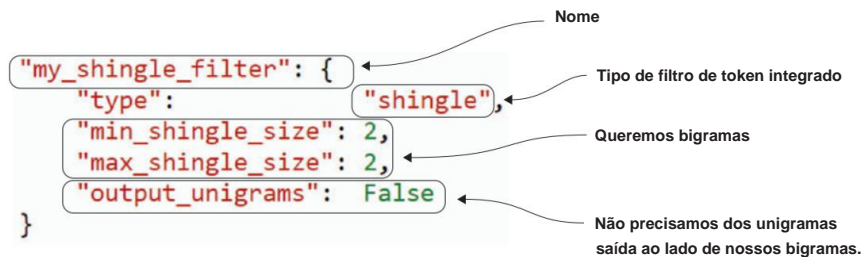
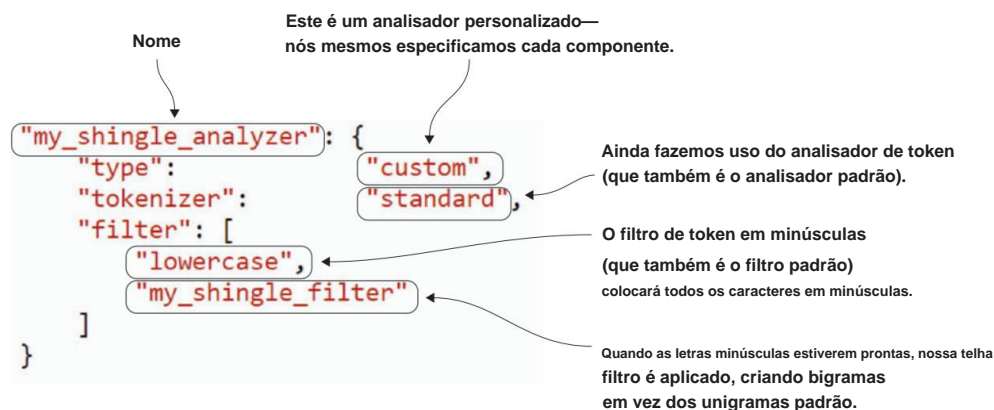


Figura 6.34 Um filtro de token shingle para produzir bigramas

O analisador mostrado na figura 6.35 é a combinação de todas as operações necessárias para vá do texto de entrada ao índice. Incorpora o filtro de telha, mas é muito mais do que esse. O tokenizer divide o texto em tokens ou termos; você pode então usar letras minúsculas filtro para que não haja diferença ao pesquisar por "Diabetes" versus "diabetes". Finalmente, você aplica seu filtro de telha, criando seus bigramas.



**Figura 6.35 Um analisador personalizado com tokenização padrão e um filtro de token shingle para produzir bigramas**

Observe que você precisa fechar o índice antes de atualizar as configurações. Você pode então reabra o índice com segurança sabendo que suas configurações foram atualizadas. Nem todas as configurações as mudanças exigem que o índice seja fechado, mas este exige. Você pode encontrar uma visão geral de quais configurações precisam que o índice seja fechado em <http://www.elastic.co/guide/en/elasticsearch/reference/current/indices-update-settings.html>.

O índice agora está pronto para usar seu novo analisador. Para isso você criará um novo documento tipo de tratamento, doenças2, com novo mapeamento, conforme listagem a seguir.

#### Listagem 6.6 Crie mapeamento de tipo de documento Elasticsearch mais avançado

```

docType = 'doenças2'
mapeamento de doença = {
  'propriedades': {
    'nome': {'tipo': 'string'},
    'título': {'tipo': 'string'},
    'texto completo': {
      "tipo": "string",
      "Campos": {
        "cobreiro": {
          "type": "corda",
          "analisador": "my_shingle_analyzer"
        }
      }
    }
  }
}

```

O novo mapeamento da  
doença difere do antigo pela  
adição do  
campo fulltext.shingles  
que contém seus bigramas.

```
    }
  }
}

client.indices.put_mapping(index=nomedoíndice,
doc_type=docType,body=diseaseMapping )
```

Dentro do texto completo agora você tem um parâmetro extra, campos. Aqui você pode especificar todos os diferentes isótopos do texto completo. Você tem apenas um; atende pelo nome de telhas e analisará o texto completo com seu novo `my_shingle_analyzer`. Você ainda tem acesso ao seu texto completo original, e você não especificou um analisador para isso, então o padrão será usado como antes. Você pode acessar o novo cedendo a propriedade nome seguido do nome do campo: `fulltext.shingles`. Tudo que você precisa fazer agora é ir siga as etapas anteriores e indexe os dados usando a API da Wikipedia, conforme mostrado em a seguinte listagem.

Listagem 6.7 Reindexando explicações de doenças da Wikipédia com novo mapeamento de tipo de documento

```
dl = wikipedia.page("Listas_de_doenças")
doençaListArray = []
para link em dl.links[15:42]:
    tentar:
        doençaListArray.append(wikipedia.page(link))
    exceto exceção, e: print str(e)

lista de verificação = [["0","1","2","3","4","5","6","7","8","9"],
["A","B","C","D","E","F","G"],
["H","I","J","K","L","M","N"],
["O","P","Q","R","S","T","U"],
["V","W","X","Y","Z"]]

para lista de doençasNumber, lista de doenças em enumerar(diseaseListArray):
    para doenças em Diseaselist.links: #loop através de listas de links para cada
        lista de doenças
            tentar:
                se doença[0] em checkList[diseaselistNumber]
e doença[0:3] != "Lista":
                páginaatual = wikipedia.page(doença)
                cliente.index(index=nomedoíndice,
doc_type=docType,id = doença, body={"nome": doença, "título":currentPage.title ,
"texto completo": currentPage.content}))
            exceto Exceção,e:
                imprimir string(e)
```

A lista de verificação é uma matriz contendo os "primeiros caracteres" permitidos. Se uma doença não cumpre, você pula.

Primeiro verifique se é uma doença e depois indexe.

Laço através doença listas.

Não há nada de novo aqui, só que desta vez você indexará `doc_type doenças2` em vez de `doenças`. Quando isso estiver concluído, você poderá avançar novamente para a etapa 4, exploração de dados, e verificar os resultados.

### 6.2.5 Passo 4 revisitado: Exploração de dados para definição de perfis de doenças

Você chegou mais uma vez à exploração de dados. Você pode adaptar a consulta de agregações e use seu novo campo para fornecer conceitos-chave relacionados ao diabetes:

Listagem 6.8 Agregação de termos significativos sobre “diabetes” com bigramas

```
corpopesquisa={
  "campos":["nome"],
  "consulta":{
    "filtrado": {
      "filtro": {
        'termo': {'nome':'diabetes'}
      }
    }
  },
  "agregações": {
    "DiseaseKeywords": {
      "significant_terms": { "campo": "texto completo", "tamanho": 30 }
    },
    "DoençaBigrams": {
      "significant_terms": "tamanho": { "campo": "texto completo.shingles",
30 }
    }
  }
}

client.search(index=indexName,doc_type=docType, body=searchBody,
from_= 0, size=3)
```

Seu novo agregado, chamado DiseaseBigrams, usa o campo `fulltext.shingles` para fornecer alguns novos insights sobre diabetes. Estes novos termos-chave aparecem:

• *Corrimento excessivo* – Um paciente com diabetes precisa urinar com frequência.

• *Causa poliúria* – Isso indica a mesma coisa: o diabetes faz com que o paciente urinar com frequência.

• *Teste de privação* – Este é na verdade um trígama, “teste de privação de água”, mas reconhece o teste de privação porque você só tem bigramas. É um teste para determinar se um paciente tem diabetes.

• *Sede excessiva* — Você já encontrou “sede” em sua pesquisa por palavra-chave unigram, mas tecnicamente, naquele ponto, poderia significar “sem sede”.

Existem outros bigramas, unigramas e provavelmente também trigramas interessantes. Tomado como um todo, podem ser usados para analisar um texto ou uma coleção de textos antes da leitura deles. Observe que você alcançou os resultados desejados sem passar pela modelagem estágio. Às vezes há pelo menos uma quantidade igual de informações valiosas a serem encontrado na exploração de dados como na modelagem de dados. Agora que você alcançou totalmente o seu Como objetivo secundário, você pode passar para a etapa 6 do processo de ciência de dados: apresentação e automação.

### 6.2.6 Passo 6: Apresentação e automação

Seu objetivo principal, o diagnóstico de doenças, se transformou em uma ferramenta de diagnóstico de autoatendimento permitindo que um médico consulte, por exemplo, um aplicativo da web. Você não vai construir neste caso, um site, mas se você planeja fazê-lo, leia a barra lateral “Elastic-search para aplicativos da web”.

#### **Elasticsearch para aplicações web** Assim

como acontece com qualquer outro banco de dados, é uma má prática expor sua API REST do Elasticsearch diretamente para o front-end de aplicativos da web. Se um site pode fazer POST diretamente solicitações ao seu banco de dados, qualquer pessoa pode excluir seus dados com a mesma facilidade: sempre há necessidade de uma camada intermediária. Essa camada intermediária pode ser Python, se for conveniente para você. Duas soluções Python populares seriam Django ou a estrutura REST Django em combinação com um front-end independente. Django geralmente é usado para construir ida e volta aplicativos (aplicativos da web onde o servidor constrói o front-end dinamicamente, dados do banco de dados e um sistema de modelos). O framework REST do Django é um plugin para o Django, transformando o Django em um serviço REST, permitindo-lhe tornar-se parte de aplicativos de página única. Um aplicativo de página única é um aplicativo da web que usa uma única página da web como âncora, mas é capaz de alterar dinamicamente o conteúdo recuperando arquivos estáticos do servidor HTTP e dados de APIs RESTful. Ambos abordagens (ida e volta e página única) são adequadas, desde que o servidor Elasticsearch em si não está aberto ao público porque não possui medidas de segurança integradas. A segurança pode ser adicionado ao Elasticsearch diretamente usando o “Shield”, um serviço pago do Elasticsearch.

O objetivo secundário, o perfil da doença, também pode ser levado ao nível de um usuário interface; é possível deixar os resultados da pesquisa produzirem uma nuvem de palavras que visualmente resume os resultados da pesquisa. Não iremos tão longe neste livro, mas se você estiver interessado em configurar algo assim em Python, use a biblioteca `word_cloud` (pip instalar `word_cloud`). Ou se você preferir JavaScript, D3.js é uma boa opção. Você pode encontrar um exemplo de implementação em <http://www.jasondavies.com/wordcloud/#%2F%2Fwww.jasondavies.com%2Fwordcloud%2Fabout%2F>.

Adicionar suas palavras-chave neste site baseado em D3.js produzirá uma palavra unigrama nuvem como a mostrada na figura 6.36 que pode ser incorporada na apresentação

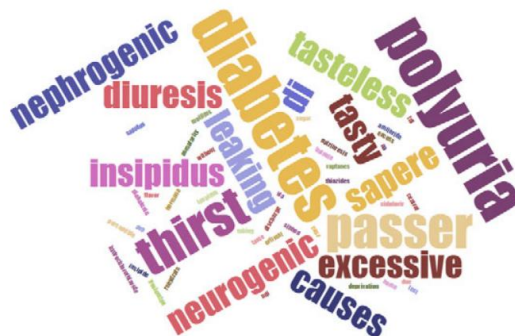


Figura 6.36 Nuvem de palavras  
Unigram em palavras-chave  
não ponderadas para diabetes do Elasticsearch

dos resultados do seu projeto. Os termos não são ponderados pela pontuação neste caso, mas já fornecem uma boa representação das descobertas.

Muitas melhorias são possíveis para sua aplicação, principalmente na área de preparação de dados. Mas mergulhar em todas as possibilidades aqui nos levaria longe demais; assim chegamos ao final deste capítulo. No próximo, daremos uma olhada no streaming de dados.

## 6.3 Resumo

Neste capítulo, você aprendeu o seguinte:

- *NoSQL* significa “Not Only Structured Query Language” e surgiu da necessidade de lidar com quantidades e variedades de dados cada vez maiores, bem como da necessidade crescente de esquemas mais diversos e flexíveis, como redes e estruturas hierárquicas.

- O tratamento de todos esses dados requer particionamento do banco de dados porque nenhuma máquina é capaz de realizar todo o trabalho. Ao particionar, aplica-se o Teorema CAP : você pode ter disponibilidade ou consistência, mas nunca as duas ao mesmo tempo. • Os bancos de dados relacionais e os bancos de dados gráficos seguem os princípios ACID : atomicidade, consistência, isolamento e durabilidade. Os bancos de dados NoSQL geralmente seguem os princípios BASE : disponibilidade básica, estado suave e consistência eventual.

- Os quatro maiores tipos de bancos de dados NoSQL –

- Armazenamentos *de valores-chave* – Essencialmente, um conjunto de pares de valores-chave armazenados em um banco de dados.

- Esses bancos de dados podem ser imensamente grandes e versáteis, mas a complexidade dos dados é baixa. Um exemplo bem conhecido é o Redis.

- *Bancos de dados de colunas largas* – Esses bancos de dados são um pouco mais complexos do que os armazenamentos de valores-chave, pois usam colunas, mas de uma forma mais eficiente do que um RDBMS normal faria. As colunas são essencialmente desacopladas, permitindo recuperar dados em uma única coluna rapidamente. Um banco de dados bem conhecido é o Cassandra.

- Armazenamentos *de documentos* – Esses bancos de dados são um pouco mais complexos e armazenam dados como documentos. Atualmente o mais popular é o MongoDB, mas em nosso estudo de caso usamos o Elasticsearch, que é ao mesmo tempo um armazenamento de documentos e um mecanismo de busca.

- *Bancos de dados gráficos* – Esses bancos de dados podem armazenar as estruturas de dados mais complexas, pois tratam as entidades e as relações entre entidades com igual cuidado.

- Essa complexidade tem um custo na velocidade de pesquisa. Um popular é o Neo4j, mas o GraphX (um banco de dados gráfico relacionado ao Apache Spark) está ganhando terreno. • Elasticsearch é um mecanismo de pesquisa de texto completo e armazenamento de documentos construído sobre o Apache Lucene, o mecanismo de pesquisa de código aberto. Ele pode ser usado para tokenizar, realizar consultas de agregação, realizar consultas dimensionais (facetadas), consultas de pesquisa de perfil e muito mais.