

# Tham gia phong trào NoSQL

## **Chương này bao gồm**

- Hiểu cơ sở dữ liệu NoSQL và tại sao chúng được sử dụng ngày nay
- Xác định sự khác biệt giữa NoSQL và các cơ sở dữ liệu quan hệ
- Xác định nguyên tắc ACID và cách nó liên quan đến nguyên tắc NoSQL BASE
- Tìm hiểu lý do tại sao định lý CAP lại quan trọng đối với việc thiết lập cơ sở dữ liệu nhiều nút
- Áp dụng quy trình khoa học dữ liệu vào dự án với cơ sở dữ liệu NoSQL Elasticsearch

Chương này được chia thành hai phần: lý thuyết và thực hành.

- Trong phần đầu tiên của chương này, chúng ta sẽ tìm hiểu cơ sở dữ liệu NoSQL nói chung và trả lời những câu hỏi sau: Tại sao chúng tồn tại? Tại sao không cho đến gần đây? Có những loại nào và tại sao bạn nên quan tâm?
- Trong phần hai, chúng ta sẽ giải quyết một vấn đề thực tế—chẩn đoán bệnh và lập hồ sơ — sử dụng dữ liệu có sẵn miễn phí, Python và cơ sở dữ liệu NoSQL.

Chắc chắn bạn đã nghe nói về cơ sở dữ liệu NoSQL và cách chúng được nhiều công ty công nghệ cao sử dụng một cách thường xuyên. Nhưng cơ sở dữ liệu NoSQL là gì và điều gì khiến chúng khác biệt so với cơ sở dữ liệu quan hệ hoặc SQL mà bạn đã quen sử dụng? *NoSQL* là viết tắt của *Not Only Structured Query Language* - *Không chỉ ngôn ngữ truy vấn có cấu trúc*, nhưng mặc dù đúng là cơ sở dữ liệu NoSQL có thể cho phép bạn truy vấn chúng bằng SQL, bạn không cần phải tập trung vào tên thực. Nhiều cuộc tranh luận đã nổ ra về cái tên này và liệu nhóm cơ sở dữ liệu mới này có nên có một cái tên chung hay không. Thay vào đó, hãy nhìn vào những gì chúng đại diện trái ngược với *relational database management systems* - *hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS)*. Cơ sở dữ liệu truyền thống nằm trên một máy tính hoặc máy chủ. Điều này đã từng tốt miễn là dữ liệu của bạn không vượt quá máy chủ, nhưng nó đã không còn đúng với nhiều công ty trong một thời gian dài. Với sự phát triển của internet, các công ty như Google và Amazon cảm thấy họ bị kìm hãm bởi các cơ sở dữ liệu một nút này và đã tìm kiếm các giải pháp thay thế.

Nhiều công ty sử dụng cơ sở dữ liệu NoSQL một nút như MongoDB vì họ muốn lược đồ linh hoạt hoặc khả năng tổng hợp dữ liệu theo cấp bậc. Dưới đây là một số ví dụ ban đầu:

- Giải pháp NoSQL đầu tiên của Google là Google BigTable, đánh dấu sự khởi đầu của *columnar databases* - *cơ sở dữ liệu cột*.<sup>1</sup>
- Amazon đã đưa ra Dynamo, a key-value store - *một cửa hàng khóa-giá trị*.<sup>2</sup>
- Hai loại cơ sở dữ liệu khác xuất hiện trong nhiệm vụ phân vùng: *document store* - *cửa hàng tài liệu* và *graph database* - *cơ sở dữ liệu đồ thị*.

Chúng ta sẽ đi vào chi tiết về từng loại trong số bốn loại sau trong chương này.

Xin lưu ý rằng, mặc dù kích thước là một yếu tố quan trọng, những cơ sở dữ liệu này không chỉ bắt nguồn từ nhu cầu xử lý khối lượng dữ liệu lớn hơn. Mọi *V* của dữ liệu lớn có ảnh hưởng (volume, variety, velocity, and sometimes veracity - khối lượng, sự đa dạng, tốc độ và đôi khi là tính xác thực). Ví dụ, cơ sở dữ liệu đồ thị có thể xử lý dữ liệu mạng. Những người đam mê cơ sở dữ liệu đồ thị thậm chí còn tuyên bố rằng mọi thứ có thể được xem như một mạng lưới. Ví dụ, làm thế nào để bạn chuẩn bị bữa tối? Với các nguyên liệu. Những nguyên liệu này được kết hợp với nhau để tạo thành món ăn và có thể được sử dụng cùng với các nguyên liệu khác để tạo thành các món ăn khác. Nhìn từ quan điểm này, nguyên liệu và công thức nấu ăn là một phần của mạng lưới. Nhưng các công thức và nguyên liệu cũng có thể được lưu trữ trong cơ sở dữ liệu quan hệ của bạn hoặc kho lưu trữ tài liệu; tất cả là ở cách bạn nhìn nhận vấn đề. Đây là sức mạnh của NoSQL: khả năng xem xét một vấn đề từ một góc độ khác, định hình cấu trúc dữ liệu cho trường hợp sử dụng. Là một nhà khoa học dữ liệu, công việc của bạn là tìm ra câu trả lời tốt nhất cho mọi vấn đề. Mặc dù đôi khi điều này vẫn dễ đạt được hơn bằng cách sử dụng RDBMS, thường thì một cơ sở dữ liệu NoSQL cụ thể sẽ cung cấp một cách tiếp cận tốt hơn.

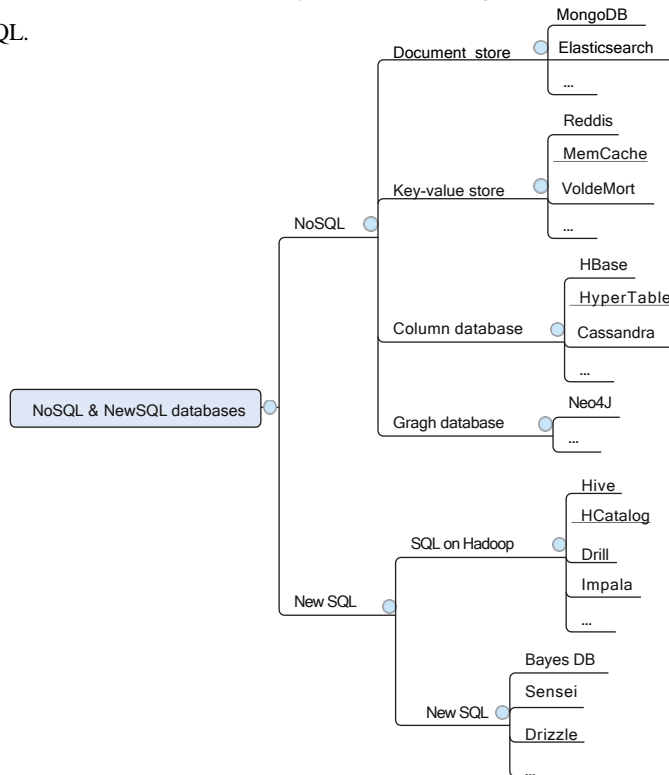
Có phải cơ sở dữ liệu quan hệ sẽ biến mất trong các công ty có dữ liệu lớn vì nhu cầu phân vùng? Không, các nền tảng NewSQL (đừng nhầm với NoSQL) là câu trả lời của RDBMS cho nhu cầu thiết lập cụm. Cơ sở dữ liệu NewSQL theo mô hình quan hệ nhưng có khả năng chia thành cụm phân tán như các cơ sở dữ liệu NoSQL. Nó không phải là dấu chấm hết cho cơ sở dữ liệu quan

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

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

hệ và chắc chắn không phải là dấu chấm hết cho SQL, vì các nền tảng như Hive dịch SQL thành các công việc MapReduce cho Hadoop. Bên cạnh đó, không phải mọi công ty đều cần dữ liệu lớn; nhiều công ty làm tốt với các cơ sở dữ liệu nhỏ và cơ sở dữ liệu quan hệ truyền thống đều hoàn hảo cho điều đó.

Nếu bạn nhìn vào sơ đồ tư duy dữ liệu lớn trong hình 6.1, bạn sẽ thấy bốn loại cơ sở dữ liệu NoSQL.



**Hình 6.1** Cơ sở dữ liệu NoSQL và NewSQL

Bốn loại này là kho tài liệu, kho khóa-giá trị, cơ sở dữ liệu đồ thị và cơ sở dữ liệu cột. Bản đồ tư duy cũng bao gồm các cơ sở dữ liệu quan hệ được phân vùng NewSQL. Trong tương lai, sự phân chia lớn này giữa NoSQL và NewSQL sẽ trở nên lỗi thời vì mọi loại cơ sở dữ liệu sẽ có tiêu điểm riêng, đồng thời kết hợp các yếu tố từ cả cơ sở dữ liệu NoSQL và NewSQL. Các dòng đang dần mờ đi khi các loại RDBMS có các tính năng NoSQL, chẳng hạn như lập chỉ mục theo hướng cột được thấy trong cơ sở dữ liệu cột. Nhưng hiện tại, đó là một cách hay để chỉ ra rằng các cơ sở dữ liệu quan hệ cũ đã vượt qua thiết lập nút đơn của chúng, trong khi các loại cơ sở dữ liệu khác đang xuất hiện dưới mẫu số NoSQL.

Hãy xem những gì NoSQL mang đến cho bảng.

## 6.1 Giới thiệu về NoSQL

Như bạn đã đọc, mục tiêu của cơ sở dữ liệu NoSQL không chỉ là cung cấp cách phân vùng cơ sở dữ liệu thành công trên nhiều nút, mà còn trình bày các cách cơ bản khác nhau để lập mô hình dữ liệu phù hợp với cấu trúc của nó đối với trường hợp sử dụng chứ không phải cách một cơ sở dữ liệu quan hệ yêu cầu nó được mô hình hóa.

Để giúp bạn hiểu về NoSQL, chúng tôi sẽ bắt đầu bằng cách xem xét các nguyên tắc ACID cốt lõi của cơ sở dữ liệu quan hệ một máy chủ và chỉ ra cách cơ sở dữ liệu NoSQL viết lại chúng thành các nguyên tắc BASE để chúng hoạt động tốt hơn nhiều theo kiểu phân tán.

Chúng ta cũng sẽ xem xét định lý CAP, mô tả vấn đề chính với việc phân phối cơ sở dữ liệu trên nhiều nút và cách cơ sở dữ liệu ACID và BASE tiếp cận nó.

### 6.1.1 ACID: nguyên tắc cốt lõi của cơ sở dữ liệu quan hệ

Các khía cạnh chính của cơ sở dữ liệu quan hệ truyền thống có thể được tóm tắt bằng khái niệm ACID:

- *Atomicity* - *nguyên tử*—Nguyên tắc “tất cả hoặc không có gì”. Nếu một bản ghi được đưa vào cơ sở dữ liệu, nó sẽ được đưa vào hoàn toàn hoặc hoàn toàn không. Ví dụ, nếu mất điện xảy ra ở giữa hành động ghi cơ sở dữ liệu, bạn sẽ không kết thúc với một nửa bản ghi; nó sẽ không ở đó chút nào.
- *Consistency* - *tính nhất quán*—Nguyên tắc quan trọng này duy trì tính toàn vẹn của dữ liệu. Không có mục nào đưa nó vào cơ sở dữ liệu sẽ xung đột với các quy tắc được xác định trước, chẳng hạn như thiếu trường bắt buộc hoặc trường ở dạng số thay vì văn bản.
- *Isolation* - *sự cách ly*—Khi một cái gì đó được thay đổi trong cơ sở dữ liệu, không có gì có thể xảy ra trên cùng một dữ liệu chính xác này vào cùng một thời điểm. Thay vào đó, các hành động xảy ra nối tiếp với các thay đổi khác. Cách ly là thang đi từ cách ly thấp đến cách ly cao. Ở quy mô này, cơ sở dữ liệu truyền thống đang ở mức “cô lập cao”. Một ví dụ về sự cô lập thấp sẽ là Google Docs: Nhiều người có thể viết vào một tài liệu cùng một lúc và thấy những thay đổi của nhau xảy ra ngay lập tức. Mặt khác, một tài liệu Word truyền thống có tính cô lập cao; nó bị khóa để chỉnh sửa bởi người dùng đầu tiên mở nó. Người thứ hai mở tài liệu có thể xem phiên bản đã lưu cuối cùng nhưng không thể xem các thay đổi chưa được lưu hoặc chỉnh sửa tài liệu mà không lưu nó dưới dạng bản sao trước. Vì vậy, một khi ai đó đã mở nó, phiên bản cập nhật nhất được cách ly hoàn toàn với tất cả trừ người đã khóa tài liệu.
- *Durability* - *độ bền*—Nếu dữ liệu đã vào cơ sở dữ liệu, nó sẽ tồn tại vĩnh viễn. Thiệt hại vật lý đối với đĩa cứng sẽ phá hủy các bản ghi, nhưng mất điện và sự cố phần mềm thì không.

ACID áp dụng cho tất cả các cơ sở dữ liệu quan hệ và một số cơ sở dữ liệu NoSQL nhất định, chẳng hạn như cơ sở dữ liệu đồ thị Neo4j. Chúng ta sẽ thảo luận thêm về cơ sở dữ liệu đồ thị sau trong chương này và trong chương 7. Đối với hầu hết các cơ sở dữ liệu NoSQL khác, một nguyên tắc khác được áp dụng: BASE - CƠ SỞ. Để hiểu BASE và tại sao nó áp dụng cho hầu hết các cơ sở dữ liệu NoSQL, chúng ta cần xem Cap Theorem - Định lý CAP.

### 6.1.2 Định lý CAP: vấn đề với DBs trên nhiều nút

Khi cơ sở dữ liệu được trải rộng trên các máy chủ khác nhau, rất khó để tuân theo nguyên tắc ACID vì tính nhất quán của ACID hứa hẹn; Định lý CAP chỉ ra lý do tại sao điều này trở thành vấn đề. Định lý CAP phát biểu rằng một cơ sở dữ liệu có thể là bất kỳ hai trong số những điều sau đây nhưng không bao giờ là cả ba:

- *Partition tolerant - chịu phân vùng*—Cơ sở dữ liệu có thể xử lý phân vùng mạng hoặc lỗi mạng.
- *Available - có sẵn*—Miễn là nút mà bạn đang kết nối đang hoạt động và bạn có thể kết nối với nút đó, thì nút đó sẽ phản hồi, ngay cả khi kết nối giữa các nút cơ sở dữ liệu khác nhau bị mất.
- *Consistent - nhất quán*—Bất kể bạn kết nối với nút nào, bạn sẽ luôn thấy cùng một dữ liệu.

Đối với cơ sở dữ liệu một nút, thật dễ dàng để thấy nó luôn sẵn có và nhất quán như thế nào:

- *Available - có sẵn*—Miễn là nút lên, nó có sẵn. Đó là tất cả những lời hứa về tính khả dụng của CAP.
- *Consistent - nhất quán*—Không có nút thứ hai, vì vậy không có gì có thể không nhất quán.

Mọi thứ trở nên thú vị khi cơ sở dữ liệu được phân vùng. Khi đó, bạn cần đưa ra lựa chọn giữa tính khả dụng và tính nhất quán, như thể hiện trong hình 6.2.

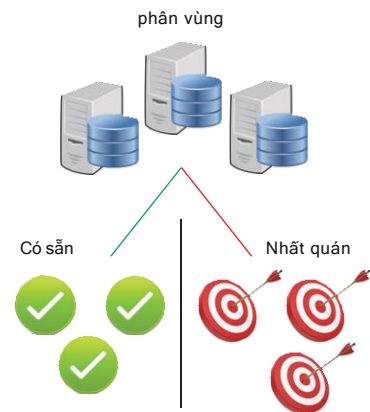
Hãy lấy ví dụ về một cửa hàng trực tuyến có máy chủ ở Châu Âu và máy chủ ở Hoa Kỳ, với một trung tâm phân phối duy nhất. Một người Đức tên Fritz và một người Mỹ tên Freddy đang mua sắm cùng một lúc trên cùng một cửa hàng trực tuyến. Họ nhìn thấy một món đồ và chỉ còn một món đồ trong kho: một chiếc bàn cà phê hình con bạch tuộc bằng đồng.

Thảm họa xảy ra và kết nối giữa hai máy chủ cục bộ tạm thời ngừng hoạt động. Nếu bạn là chủ cửa hàng, bạn sẽ có hai lựa chọn:

- *Availability - khả dụng*—Bạn cho phép các máy chủ tiếp tục phục vụ khách hàng và bạn sắp xếp mọi thứ sau đó.
- *Consistency - tính nhất quán*—Bạn tạm dừng tất cả hoạt động bán hàng cho đến khi kết nối được thiết lập lại.

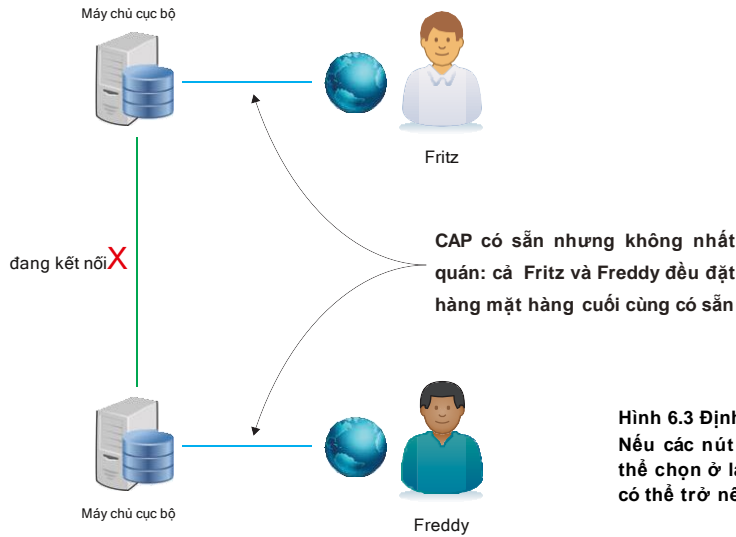
Trong trường hợp đầu tiên, cả Fritz và Freddy sẽ mua bàn cà phê bạch tuộc, bởi vì số lượng hàng tồn kho được biết đến cuối cùng cho cả hai nút là “một” và cả hai nút đều được phép bán nó, như thể hiện trong hình 6.3.

Nếu bàn cà phê khó kiếm, bạn sẽ phải thông báo cho Fritz hoặc Freddy rằng anh ta sẽ không nhận được bàn của mình vào ngày giao hàng đã hứa hoặc tệ hơn nữa là anh ta sẽ



**Hình 6.2 Định lý CAP**

Khi phân vùng cơ sở dữ liệu của bạn, bạn cần chọn giữa tính khả dụng và tính nhất quán.



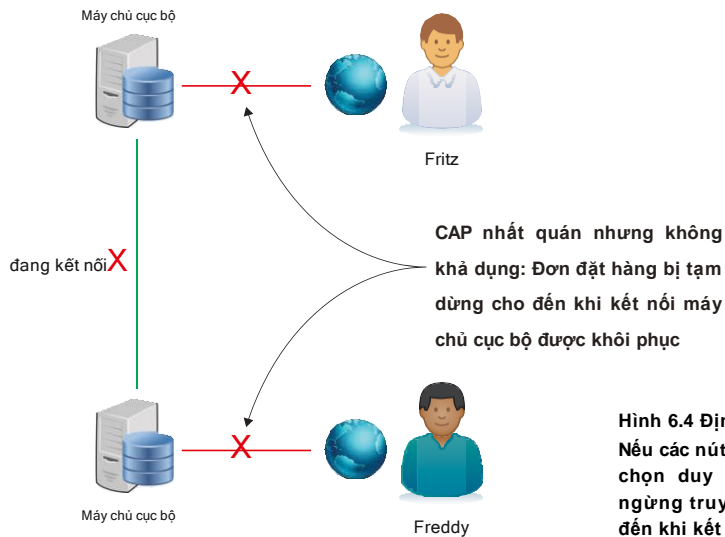
Hình 6.3 Định lý CAP

Nếu các nút bị ngắt kết nối, bạn có thể chọn ở lại có sẵn, nhưng dữ liệu có thể trở nên không nhất quán.

không bao giờ nhận được nó. Là một doanh nhân giỏi, bạn có thể đền bù cho một trong số họ bằng một phiếu giảm giá cho lần mua hàng sau, và mọi chuyện có thể sẽ ổn thỏa sau đó.

Tùy chọn thứ hai (hình 6.4) liên quan đến việc tạm thời tạm dừng các yêu cầu gửi đến.

Điều này có thể công bằng cho cả Fritz và Freddy nếu sau năm phút cửa hàng trực tuyến mở cửa kinh doanh trở lại, nhưng sau đó bạn có thể mất cả doanh số bán hàng và có thể nhiều hơn nữa. Các cửa hàng trực tuyến có xu hướng chọn tính sẵn có hơn tính nhất quán, nhưng đó không phải là lựa chọn tối ưu trong tất cả các trường hợp.



Hình 6.4 Định lý CAP

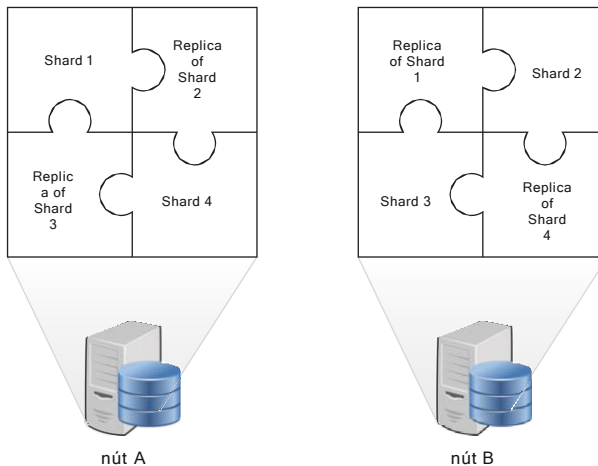
Nếu các nút bị ngắt kết nối, bạn có thể chọn duy trì nhất quán bằng cách ngừng truy cập vào cơ sở dữ liệu cho đến khi kết nối được khôi phục

Tham gia một lễ hội nổi tiếng như Tomorrowland. Lễ hội có xu hướng có sức chứa tối đa cho phép vì lý do an toàn. Nếu bạn bán nhiều vé hơn mức cho phép vì máy chủ của bạn tiếp tục bán trong khi lỗi liên lạc nút, bạn có thể bán gấp đôi số lượng cho phép vào thời điểm liên lạc được thiết lập lại. Trong trường hợp như vậy, có thể khôn ngoan hơn nếu bạn nhất quán và tạm thời tắt các nút. Dù sao thì một lễ hội như Tomorrowland cũng được bán hết vé trong vài giờ đầu tiên, vì vậy một chút thời gian ngừng hoạt động sẽ không ảnh hưởng nhiều bằng việc phải rút hàng nghìn vé vào cửa.

### 6.1.3 Các nguyên tắc CƠ BẢN của cơ sở dữ liệu NoSQL

RDBMS tuân theo các nguyên tắc ACID; Cơ sở dữ liệu NoSQL không tuân theo ACID, chẳng hạn như kho lưu trữ tài liệu và kho lưu trữ khóa-giá trị, mà sẽ tuân theo BASE. BASE là một tập hợp các nguyên tắc cơ sở dữ liệu nhẹ nhàng hơn nhiều:

- *Basically available - cơ bản có sẵn*—Tính khả dụng được đảm bảo theo nghĩa CAP. Lấy ví dụ về cửa hàng trực tuyến, nếu một nút đang hoạt động, bạn có thể tiếp tục mua sắm. Tùy thuộc vào cách mọi thứ được thiết lập, các nút có thể tiếp quản từ các nút khác. Ví dụ, Elasticsearch là một công cụ tìm kiếm loại tài liệu NoSQL phân chia và sao chép dữ liệu của nó theo cách mà lỗi nút không nhất thiết có nghĩa là lỗi dịch vụ, thông qua quá trình *sharding*. Mỗi shard – *phân đoạn* có thể được coi là một phiên bản máy chủ cơ sở dữ liệu riêng lẻ, nhưng cũng có khả năng giao tiếp với các phân đoạn khác để phân chia khối lượng công việc một cách hiệu quả nhất có thể (hình 6.5). Một số phân đoạn có thể có mặt trên một nút. Nếu mỗi phân đoạn có một bản sao trên một nút khác, lỗi nút có thể dễ dàng khắc phục bằng cách phân chia lại công việc cho các nút còn lại.
- *Soft state - trạng thái mềm*—Trạng thái của một hệ thống có thể thay đổi theo thời gian. Điều này tương ứng với *nguyên tắc nhất quán cuối cùng*: hệ thống có thể phải thay đổi để tạo dữ liệu nhất quán trở lại. Trong một nút, dữ liệu có thể là “A” và ở nút kia, dữ liệu có

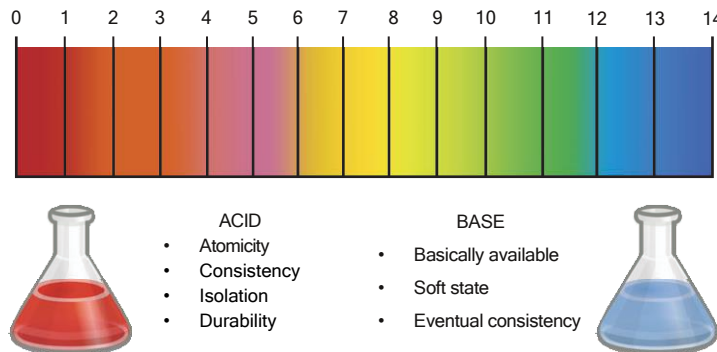


Hình 6.5 Phân đoạn: mỗi phân đoạn có thể hoạt động như một cơ sở dữ liệu độc lập, nhưng chúng cũng hoạt động cùng nhau như một tổng thể. Ví dụ này đại diện cho hai nút, mỗi nút chứa bốn phân đoạn: hai phân đoạn chính và hai bản sao. Lỗi của một nút được sao lưu bởi nút kia.

- thể là “B” vì nó đã được điều chỉnh. Sau đó, lúc giải quyết xung đột khi mạng trực tuyến trở lại, có thể “A” trong nút đầu tiên được thay thế bằng “B”. Mặc dù không ai làm bất cứ điều gì để thay đổi “A” thành “B” một cách rõ ràng, nhưng nó sẽ nhận giá trị này khi nó trở nên nhất quán với nút khác.
- *Eventual consistency - tính nhất quán cuối cùng*—Cơ sở dữ liệu sẽ trở nên nhất quán theo thời gian. Trong ví dụ về cửa hàng trực tuyến, chiếc bàn được bán hai lần, dẫn đến dữ liệu không nhất quán. Khi kết nối giữa các nút riêng lẻ được thiết lập lại, chúng sẽ giao tiếp và quyết định cách giải quyết. Xung đột này có thể được giải quyết, ví dụ, trên cơ sở ai đến trước được phục vụ trước hoặc bằng cách ưu tiên khách hàng chịu chi phí vận chuyển thấp nhất. Cơ sở dữ liệu đi kèm với hành vi mặc định, nhưng do có một quyết định kinh doanh thực tế cần đưa ra ở đây, hành vi này có thể bị ghi đè. Ngay cả khi kết nối được thiết lập và đang chạy, độ trễ có thể khiến các nút trở nên không nhất quán. Thông thường, các sản phẩm được giữ trong giỏ mua hàng trực tuyến nhưng việc đặt một mặt hàng vào giỏ không khóa đối với những người dùng khác. Nếu Fritz đánh bại Freddy ở nút thanh toán, sẽ có vấn đề xảy ra khi Freddy thanh toán. Điều này có thể dễ dàng giải thích cho khách hàng: anh ta đã quá muộn. Nhưng điều gì sẽ xảy ra nếu cả hai cùng nhấn nút thanh toán trong cùng một phần nghìn giây và cả hai lần bán hàng đều diễn ra?

### ACID so với BASE

Các nguyên tắc BASE phần nào được tạo ra để phù hợp với *acid* và *base* từ hóa học: acid là chất lỏng có giá trị pH thấp. Một base thì ngược lại và có giá trị pH cao. Chúng ta sẽ không đi sâu vào các chi tiết hóa học ở đây, nhưng hình 6.6 cho thấy một cách dễ nhớ đối với những thứ quen thuộc với các đương lượng hóa học của acid và base.



Hình 6.6 ACID so với BASE: cơ sở dữ liệu quan hệ truyền thống so với hầu hết các cơ sở dữ liệu NoSQL. Các tên được bắt nguồn từ khái niệm hóa học của thang đo pH. Giá trị pH dưới 7 có tính axit; cao hơn 7 là một bazơ. Trên thang đo này, nước bề mặt trung bình của bạn dao động trong khoảng từ 6,5 đến 8,5.

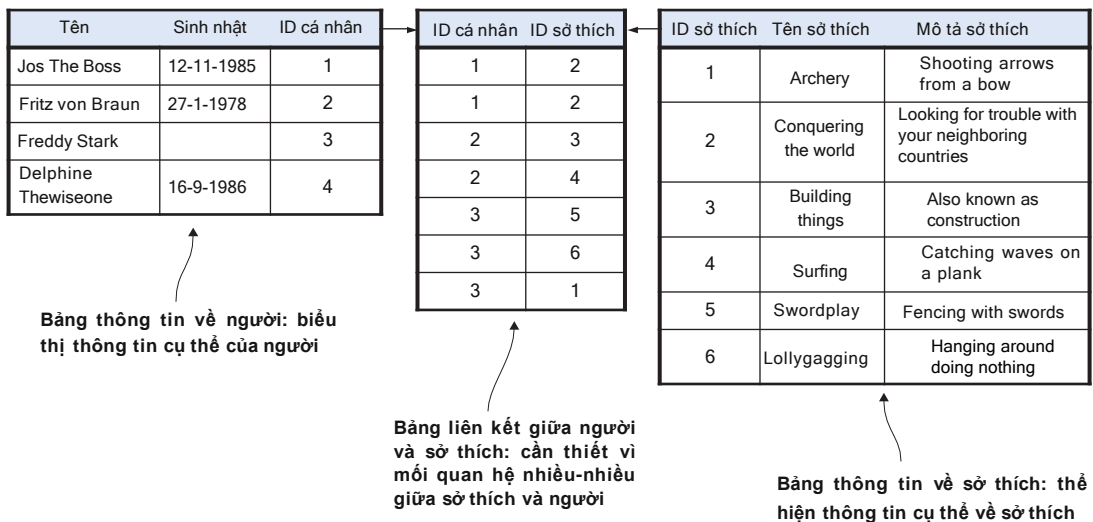


### 6.1.4 Các loại cơ sở dữ liệu NoSQL

Như bạn đã thấy trước đó, có bốn loại NoSQL lớn: kho lưu trữ khóa-giá trị, kho lưu trữ tài liệu, cơ sở dữ liệu hướng cột và cơ sở dữ liệu đồ thị. Mỗi loại giải quyết một vấn đề không thể giải quyết bằng cơ sở dữ liệu quan hệ. Việc triển khai thực tế thường là sự kết hợp của những điều này. Ví dụ, OrientDB là một *multi-model database* - cơ sở dữ liệu đa mô hình, kết hợp các loại NoSQL. OrientDB là một cơ sở dữ liệu đồ thị trong đó mỗi nút là một tài liệu.

Trước khi đi vào các cơ sở dữ liệu NoSQL khác nhau, hãy xem các cơ sở dữ liệu quan hệ để bạn có cái gì đó để so sánh chúng với nhau. Trong mô hình hóa dữ liệu, có nhiều cách để tiếp cận. Cơ sở dữ liệu quan hệ thường cố gắng hướng tới *normalization* - bình thường hóa: đảm bảo mỗi phần dữ liệu chỉ được lưu trữ một lần. Bình thường hóa đánh dấu thiết lập cấu trúc của chúng. Ví dụ: nếu bạn muốn lưu trữ dữ liệu về một người và sở thích của họ, bạn có thể làm như vậy với hai bảng: một bảng về người đó và một bảng về sở thích của họ. Như bạn có thể thấy trong hình 6.7, một bảng bổ sung là cần thiết để liên kết các sở thích với mọi người vì sở thích của họ là *many-to-many relationship* - mối quan hệ nhiều-nhiều: một người có thể có nhiều sở thích và một sở thích có thể có nhiều người cùng thực hiện.

Một cơ sở dữ liệu quan hệ quy mô đầy đủ có thể được tạo thành từ nhiều thực thể và các bảng liên kết. Bây giờ bạn đã có một số thứ để so sánh với NoSQL, hãy xem các loại khác nhau sau.



Hình 6.7 Cơ sở dữ liệu quan hệ hướng tới chuẩn hóa (đảm bảo mỗi phần dữ liệu chỉ được lưu trữ một lần). Mỗi bảng có các mã định danh duy nhất (khóa chính) được sử dụng để mô hình hóa mối quan hệ giữa các thực thể (bảng), do đó có thuật ngữ quan hệ.

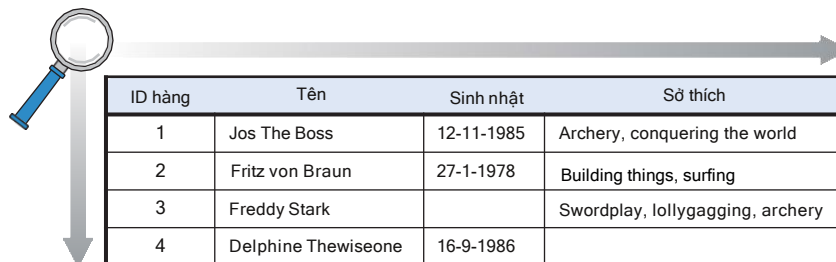
## COLUMN-ORIENTED DATABASE- CƠ SỞ DỮ LIỆU HƯỚNG CỘT

Cơ sở dữ liệu quan hệ truyền thống được định hướng theo hàng, với mỗi hàng có id hàng và mỗi trường trong hàng được lưu trữ cùng nhau trong một bảng. Chẳng hạn, giả sử rằng không có dữ liệu bổ sung nào về sở thích được lưu trữ và bạn chỉ có một bảng duy nhất để mô tả mọi người, như thể hiện trong hình 6.8. Lưu ý rằng trong tình huống này, bạn có một chút bất chuẩn hóa vì sở thích có thể được lặp lại. Nếu thông tin về sở thích là một phần bổ sung thú vị nhưng không cần thiết cho trường hợp sử dụng của bạn, thì việc thêm thông tin đó dưới dạng danh sách trong cột Sở thích là một cách tiếp cận có thể chấp nhận được. Nhưng nếu thông tin không đủ quan trọng cho một bảng riêng biệt, thì nó có nên được lưu trữ không?

ID hàng	Tên	Sinh nhật	Sở thích
1	Jos The Boss	12-11-1985	Archery, conquering the world
2	Fritz von Braun	27-1-1978	Building things, surfing
3	Freddy Stark		Swordplay, lollygagging, archery
4	Delphine Thewiseone	16-9-1986	

**Hình 6.8** Bố cục cơ sở dữ liệu theo hàng. Mỗi thực thể (người) được đại diện bởi một hàng duy nhất, trải rộng trên nhiều cột.

Mỗi khi bạn tra cứu thứ gì đó trong cơ sở dữ liệu theo hàng, mọi hàng đều được quét, bất kể bạn yêu cầu cột nào. Giả sử bạn chỉ muốn có một danh sách các ngày sinh trong tháng 9. Cơ sở dữ liệu sẽ quét bảng từ trên xuống dưới và từ trái sang phải, như thể hiện trong hình 6.9, cuối cùng trả về danh sách ngày sinh.



**Hình 6.9** Tra cứu theo hàng: từ trên xuống dưới và cứ mỗi mục nhập, tất cả các cột đều được đưa vào bộ nhớ

Việc lập chỉ mục dữ liệu trên một số cột nhất định có thể cải thiện đáng kể tốc độ tra cứu, nhưng việc lập chỉ mục cho mọi cột sẽ mang lại thêm chi phí hoạt động và cơ sở dữ liệu vẫn đang quét tất cả các cột.

Cơ sở dữ liệu cột lưu trữ từng cột riêng biệt, cho phép quét nhanh hơn khi chỉ có một số ít cột tham gia; xem hình 6.10.

Tên	ID hàng
Jos The Boss	1
Fritz von Braun	2
Freddy Stark	3
Delphine Thewiseone	4

Sinh nhật	ID hàng
12-11-1985	1
27-1-1978	2
16-9-1986	4

Sở thích	ID hàng
Archery	1, 3
Conquering the world	1
Building things	2
Surfing	2
Swordplay	3
Lollygagging	3

**Hình 6.10** Cơ sở dữ liệu hướng cột lưu trữ từng cột riêng biệt với số hàng liên quan. Mỗi thực thể (người) được chia thành nhiều bảng.

Bố cục này trông rất giống với cơ sở dữ liệu hướng hàng với chỉ mục trên mỗi cột. Một cơ sở dữ liệu *index – chỉ mục* là một cấu trúc dữ liệu cho phép tra cứu nhanh dữ liệu với chỉ phí không gian lưu trữ và ghi bổ sung (cập nhật chỉ mục). Chỉ mục ánh xạ số hàng tới dữ liệu, trong khi cơ sở dữ liệu cột ánh xạ dữ liệu tới số hàng; theo cách đó, việc đếm trở nên nhanh hơn, vì vậy thật dễ dàng để xem có bao nhiêu người thích bắn cung chẳng hạn. Việc lưu trữ các cột một cách riêng biệt cũng cho phép tối ưu hóa quá trình nén vì chỉ có một loại dữ liệu trên mỗi bảng.

Khi nào bạn nên sử dụng cơ sở dữ liệu hướng hàng và khi nào bạn nên sử dụng cơ sở dữ liệu hướng cột? Trong cơ sở dữ liệu hướng cột, thật dễ dàng để thêm một cột khác vì không cột nào hiện có bị ảnh hưởng bởi cột đó. Nhưng việc thêm toàn bộ bản ghi yêu cầu điều chỉnh tất cả các bảng. Điều này làm cho cơ sở dữ liệu hướng hàng thích hợp hơn cơ sở dữ liệu hướng cột để xử lý giao dịch trực tuyến (OLTP), bởi vì điều này ngụ ý thêm hoặc thay đổi bản ghi liên tục. Cơ sở dữ liệu hướng cột tỏa sáng khi thực hiện phân tích và báo cáo: tổng hợp các giá trị và đếm mục nhập. Cơ sở dữ liệu hướng hàng thường là cơ sở dữ liệu hoạt động được lựa chọn cho các giao dịch thực tế (chẳng hạn như bán hàng). Các tác vụ hàng loạt nhanh chóng giúp cập nhật cơ sở dữ liệu định hướng cột, hỗ trợ tra cứu và tổng hợp tốc độ cực nhanh bằng cách sử dụng thuật toán MapReduce cho các báo cáo. Các ví dụ về cửa hàng cột gia đình là Hbase, Cassandra của Facebook, Hypertable, và ông tổ của các cửa hàng nhiều cột, Google BigTable.

## KEY-VALUE STORES – KIỂU LƯU TRỮ KHÓA GIÁ TRỊ

Kho lưu trữ khóa giá trị là cơ sở dữ liệu NoSQL ít phức tạp nhất. Đúng như tên gọi, chúng là một tập hợp các cặp khóa-giá trị, như thể hiện trong hình 6.11, và sự đơn giản này khiến chúng trở thành loại cơ sở dữ liệu NoSQL có khả năng mở rộng nhất, có khả năng lưu trữ lượng dữ liệu khổng lồ.

Key	Value
Tên	Jos The Boss
Sinh nhật	12-11-1985
Sở thích	Archery, conquering the world

**Hình 6.11 Cửa hàng khóa-giá trị lưu trữ mọi thứ dưới dạng khóa và giá trị.**

Giá trị trong kho lưu trữ khóa-giá trị có thể là bất kỳ thứ gì: một chuỗi, một số nhưng cũng có thể là một tập hợp hoàn toàn mới các cặp khóa-giá trị được gói gọn trong một đối tượng. Hình 6.12 cho thấy cấu trúc khóa-giá trị phức tạp hơn một chút. Ví dụ về kho lưu trữ khóa-giá trị là Redis, Voldemort, Riak và Amazon's Dynamo.

```
{ "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 }
  ] }
] } ] }
```

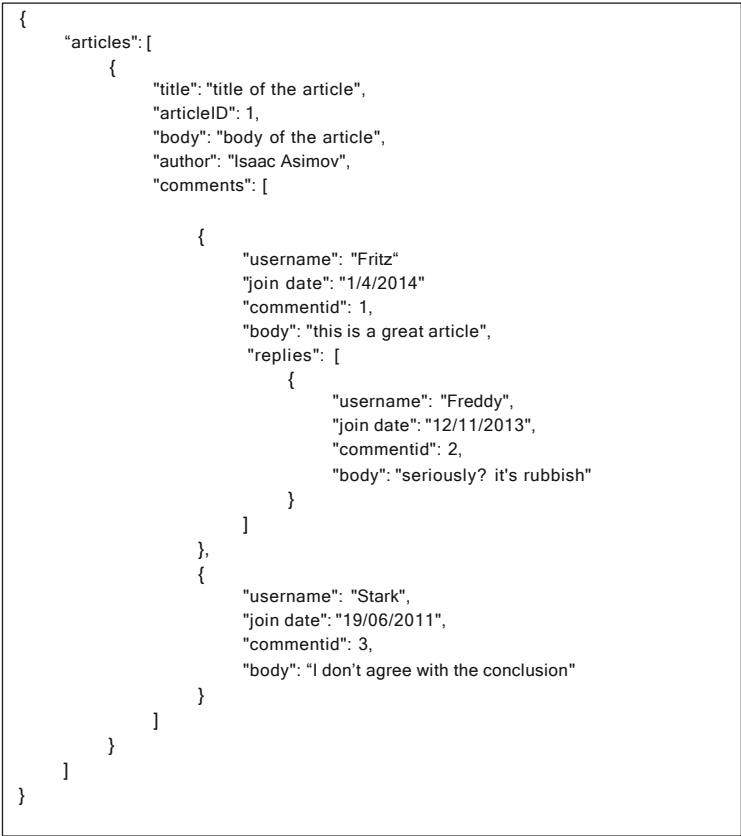
**Hình 6.12 Khóa-giá trị cấu trúc lồng nhau**

## DOCUMENT STORES – KIỂU LƯU TRỮ TÀI LIỆU

Kho lưu trữ tài liệu phức tạp hơn một bước so với kho lưu trữ khóa-giá trị: kho lưu trữ tài liệu đảm nhận một cấu trúc tài liệu nhất định có thể được chỉ định bằng lược đồ. Các cửa hàng tài liệu xuất hiện tự nhiên nhất trong số các loại cơ sở dữ liệu NoSQL vì chúng được thiết kế để lưu trữ các tài liệu hàng ngày, và chúng cho phép truy vấn và tính toán phức tạp trên dạng dữ liệu thường được tổng hợp này. Cách mọi thứ được lưu trữ trong cơ sở dữ liệu quan hệ có ý nghĩa từ quan điểm chuẩn hóa: mọi thứ chỉ được lưu trữ một lần và được kết nối thông qua khóa ngoại. Các cửa hàng tài liệu ít quan tâm đến việc chuẩn hóa miễn là dữ liệu ở trong một cấu trúc có ý nghĩa. Một mô hình dữ liệu quan hệ không phải lúc nào cũng phù hợp với các trường hợp kinh doanh nhất định. Các tờ báo hoặc tạp chí, ví dụ, chứa các bài báo. Để lưu trữ chúng trong một cơ sở dữ liệu quan hệ, trước tiên bạn cần phải cắt nhỏ chúng: văn bản bài báo nằm trong một bảng, tác giả và tất cả thông tin về tác giả trong một bảng khác, và các nhận xét về bài báo khi được xuất bản trên một trang web sẽ nằm trong một bảng khác. Như thể hiện trong hình 6.13, một bài báo cũng có thể được lưu trữ dưới dạng một thực thể duy nhất; điều



Phương pháp lưu trữ tài liệu



Hình 6.13 Kho lưu trữ tài liệu lưu toàn bộ tài liệu, trong khi RDMS cắt nhỏ bài báo và lưu nó vào một số bảng. Ví dụ được lấy từ trang web the Guardian.

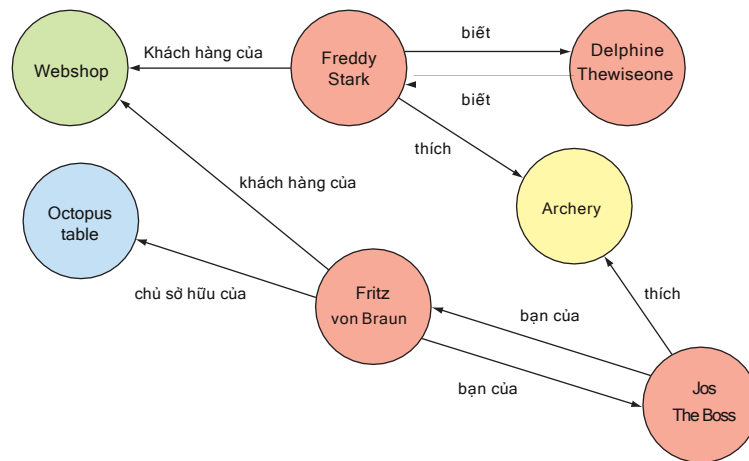
này làm giảm gánh nặng nhận thức khi làm việc với dữ liệu cho những người thường xuyên xem các bài báo. Ví dụ về kho lưu trữ tài liệu là MongoDB và CouchDB.

## GRAPH DATABASE – CSDL ĐỒ THỊ

Loại cơ sở dữ liệu NoSQL lớn cuối cùng là loại phức tạp nhất, hướng đến việc lưu trữ các mối quan hệ giữa các thực thể một cách hiệu quả. Khi dữ liệu có tính liên kết cao, chẳng hạn như đối với mạng xã hội, trích dẫn bài báo khoa học hoặc cụm tài sản vốn, cơ sở dữ liệu đồ thị là câu trả lời. Dữ liệu đồ thị hoặc mạng có hai thành phần chính:

- *Node - nút*—Bản thân các thực thể. Trong một mạng xã hội, đây có thể là con người.
- *Edge - bờ rìa*—Mối quan hệ giữa hai thực thể. Mối quan hệ này được biểu diễn bằng một đường thẳng và có những thuộc tính riêng của nó. Ví dụ, một cạnh có thể có hướng nếu mũi tên chỉ ra ai là ông chủ của ai.

Đồ thị có thể trở nên vô cùng phức tạp khi có đủ các loại thực thể và mối quan hệ. Hình 6.14 đã cho thấy sự phức tạp đó chỉ với một số thực thể hạn chế. Cơ sở dữ liệu đồ thị như Neo4j cũng tuyên bố duy trì ACID, trong khi kho lưu trữ tài liệu và kho lưu trữ giá trị khóa-giá trị tuân thủ theo BASE.



Hình 6.14 Ví dụ về dữ liệu đồ thị với bốn loại thực thể (người, sở thích, công ty và đồ nội thất) và các mối quan hệ của chúng mà không có thêm thông tin về cạnh hoặc nút

Các khả năng là vô tận và bởi vì thế giới ngày càng trở nên kết nối với nhau, cơ sở dữ liệu đồ thị có khả năng giành được lợi thế so với các loại khác, bao gồm cả cơ sở dữ liệu quan hệ vẫn đang chiếm ưu thế. Bạn có thể tìm thấy bảng xếp hạng các cơ sở dữ liệu phổ biến nhất và tiến trình của chúng tại <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

Hình 6.15 Top 15 cơ sở dữ liệu được xếp hạng theo mức độ phổ biến theo DB-Engines.com vào tháng 3 năm 2015

Hình 6.15 cho thấy rằng với 9 mục, cơ sở dữ liệu quan hệ vẫn chiếm ưu thế trong top 15 vào thời điểm cuốn sách này được viết và với sự xuất hiện của NewSQL, chúng ta chưa thể đếm hết. Neo4j, cơ sở dữ liệu đồ thị phổ biến nhất, có thể được tìm thấy ở vị trí 23 tại thời điểm viết bài, với Titan ở vị trí 53.

Bây giờ bạn đã thấy từng loại cơ sở dữ liệu NoSQL, đã đến lúc bạn bắt tay vào làm với một trong số chúng.

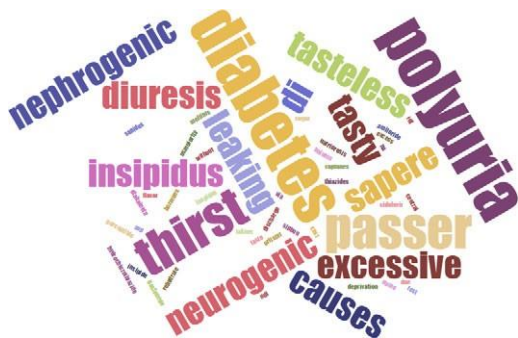
## 6.2 Case study: Tôi đang mắc bệnh gì?

Trường hợp này hẳn xảy ra với nhiều người trong chúng ta: bạn có các triệu chứng bệnh lý đột ngột và điều đầu tiên bạn làm là tra Google xem các triệu chứng đó có thể chỉ ra bệnh gì; sau đó bạn quyết định xem có đáng để gặp bác sĩ hay không. Một công cụ tìm kiếm trên web là phù hợp cho việc này, nhưng một cơ sở dữ liệu chuyên dụng hơn sẽ tốt hơn. Cơ sở dữ liệu như thế này tồn tại và khá tiên tiến; chúng có thể gần như là một phiên bản ảo của Tiến sĩ House, một bác sĩ chẩn đoán xuất sắc trong phim truyền hình *House M.D.* Nhưng chúng được xây dựng dựa trên dữ liệu được bảo vệ tốt và không phải tất cả dữ liệu đó đều có thể truy cập được bởi công chúng. Ngoài ra, mặc dù các công ty dược phẩm lớn và bệnh viện tiên tiến có quyền truy cập vào các bác sĩ ảo này, nhưng nhiều bác sĩ đa khoa vẫn mắc kẹt với sổ sách của họ. Sự bất cân xứng về thông tin và tài nguyên này không chỉ đáng buồn và nguy hiểm mà nó hoàn toàn không cần thiết. Nếu tất cả các bác sĩ đa khoa trên thế giới đều sử dụng một công cụ tìm kiếm đơn giản, dành riêng cho từng loại bệnh, thì có thể tránh được nhiều sai sót y khoa.

Trong nghiên cứu điển hình này, bạn sẽ học cách xây dựng một công cụ tìm kiếm như vậy tại đây, mặc dù chỉ sử dụng một phần nhỏ dữ liệu y tế có thể truy cập miễn phí. Để giải quyết vấn đề, bạn sẽ sử dụng cơ sở dữ liệu NoSQL hiện đại có tên là Elasticsearch để lưu trữ dữ liệu và

quy trình khoa học dữ liệu để làm việc với dữ liệu và biến nó thành một tài nguyên tìm kiếm nhanh chóng và dễ dàng. Đây là cách bạn sẽ áp dụng quy trình:

1. *Thiết lập mục tiêu nghiên cứu.*
2. *Thu thập dữ liệu*—Bạn sẽ nhận được dữ liệu của mình từ Wikipedia. Có nhiều nguồn hơn, nhưng với mục đích minh họa, một nguồn duy nhất sẽ làm được.
3. *Chuẩn bị dữ liệu*—Dữ liệu Wikipedia có thể không hoàn hảo ở định dạng hiện tại. Bạn sẽ áp dụng một số kỹ thuật để thay đổi điều này.
4. *Khám phá dữ liệu*—Trường hợp sử dụng của bạn đặc biệt ở chỗ bước 4 của quy trình khoa học dữ liệu cũng là kết quả cuối cùng mong muốn: bạn muốn dữ liệu của mình trở nên dễ khám phá.
5. *Mô hình hóa dữ liệu*—Không có mô hình dữ liệu thực nào được áp dụng trong chương này. Các ma trận thuật ngữ tài liệu được sử dụng để tìm kiếm thường là điểm khởi đầu cho việc lập mô hình chủ đề nâng cao. Chúng ta sẽ không đi sâu vào vấn đề đó ở đây.
6. *Trình bày kết quả*—Để làm cho dữ liệu có thể tìm kiếm được, bạn cần có giao diện người dùng, chẳng hạn như một trang web nơi mọi người có thể truy vấn và truy xuất thông tin về bệnh tật. Trong chương này, bạn sẽ không đi xa đến mức xây dựng một giao diện thực tế. Mục tiêu phụ của bạn: lập hồ sơ một loại bệnh theo từ khóa của nó; bạn sẽ đạt đến giai đoạn này của quy trình khoa học dữ liệu vì bạn sẽ trình bày nó dưới dạng đám mây từ, chẳng hạn như đám mây trong hình 6.16.



Hình 6.16 Một đám mây từ mẫu về các từ khóa bệnh tiểu đường không trọng số

Để tuân theo mã, bạn sẽ cần các mục sau:

- Phiên Python có cài đặt thư viện `elasticsearch-py` và Wikipedia (`pip install elasticsearch` and `pip install wikipedia`)
- Một phiên bản Elasticsearch được thiết lập cục bộ; xem phụ lục A để biết hướng dẫn cài đặt
- Thư viện IPython

**GHI CHÚ** Mã cho chương này có sẵn để tải xuống từ trang web Manning cho cuốn sách này tại <https://manning.com/books/introducing-data-science> và ở định dạng IPython.



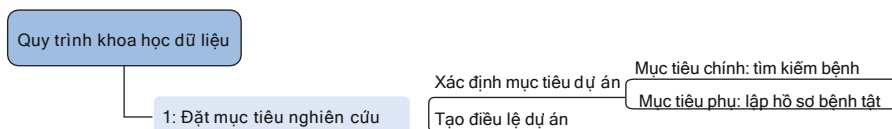
## Elasticsearch: công cụ tìm kiếm mã nguồn mở/cơ sở dữ liệu NoSQL

Để giải quyết vấn đề hiện tại, chẩn đoán bệnh, cơ sở dữ liệu NoSQL bạn sẽ sử dụng là Elasticsearch. Giống như MongoDB, Elasticsearch là một kho lưu trữ tài liệu. Nhưng không như MongoDB, Elasticsearch là một công cụ tìm kiếm. Trong khi MongoDB rất giỏi trong việc thực hiện các phép tính phức tạp và các công việc MapReduce, thì mục đích chính của Elasticsearch là tìm kiếm toàn văn bản. Elasticsearch sẽ thực hiện các phép tính cơ bản trên dữ liệu số được lập chỉ mục như tổng, số lượng, trung vị, trung bình, độ lệch chuẩn, v.v., nhưng về bản chất, nó vẫn là một công cụ tìm kiếm.

Elasticsearch được xây dựng dựa trên Apache Lucene, công cụ tìm kiếm Apache được tạo ra vào năm 1999. Lucene nổi tiếng là khó xử lý và là một khối xây dựng cho các ứng dụng thân thiện với người dùng hơn là một giải pháp cuối. Nhưng Lucene là một công cụ tìm kiếm cực kỳ mạnh mẽ và Apache Solr đã theo sau vào năm 2004, mở cửa cho công chúng sử dụng vào năm 2006. Solr (một nền tảng tìm kiếm doanh nghiệp, mã nguồn mở) được xây dựng dựa trên Apache Lucene và tại thời điểm này vẫn là công cụ linh hoạt và hiệu quả nhất. Solr là một nền tảng tuyệt vời và đáng để nghiên cứu nếu bạn tham gia vào một dự án yêu cầu công cụ tìm kiếm. Năm 2010, Elasticsearch xuất hiện và nhanh chóng trở nên phổ biến. Mặc dù Solr vẫn có thể khó thiết lập và định cấu hình, ngay cả đối với các dự án nhỏ, nhưng Elasticsearch không thể dễ dàng hơn. Solr vẫn có lợi thế trong số lượng plugin có thể mở rộng chức năng cốt lõi của nó, nhưng Elasticsearch đang nhanh chóng bắt kịp và ngày nay khả năng của nó có chất lượng tương đương.

### 6.2.1 Bước 1: Đặt mục tiêu nghiên cứu

Liệu bạn có thể chẩn đoán một căn bệnh khi kết thúc chương này mà không cần sử dụng gì ngoài chiếc máy tính ở nhà của bạn cùng với phần mềm và dữ liệu miễn phí ngoài kia không? Biết bạn muốn làm gì và làm như thế nào là bước đầu tiên trong quy trình khoa học dữ liệu, như thể hiện trong hình 6.17.



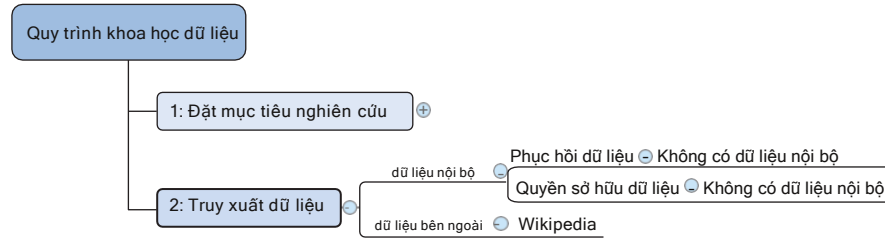
Hình 6.17 Bước 1 trong quy trình khoa học dữ liệu: đặt mục tiêu nghiên cứu

- Mục tiêu chính của bạn là thiết lập một công cụ tìm kiếm bệnh có thể giúp các bác sĩ đa khoa chẩn đoán bệnh.
- Mục tiêu phụ của bạn là lập hồ sơ bệnh: Những từ khóa nào phân biệt nó với các bệnh khác?

Mục tiêu thứ yếu này hữu ích cho mục đích giáo dục hoặc làm đầu vào cho các mục đích sử dụng nâng cao hơn, chẳng hạn như phát hiện dịch bệnh lây lan bằng cách truy cập vào phương tiện truyền thông xã hội. Với mục tiêu nghiên cứu và kế hoạch hành động đã được xác định, ta sẽ chuyển sang bước truy xuất dữ liệu.

### 6.2.2 Bước 2 và 3: Truy xuất và chuẩn bị dữ liệu

Truy xuất dữ liệu và chuẩn bị dữ liệu là hai bước riêng biệt trong quy trình khoa học dữ liệu và mặc dù điều này vẫn đúng với nghiên cứu điển hình, chúng ta sẽ khám phá cả hai trong cùng một phần. Bằng cách này, bạn có thể tránh thiết lập bộ nhớ trung gian cục bộ và ngay lập tức chuẩn bị dữ liệu trong khi dữ liệu đang được truy xuất. Hãy xem chúng ta đang ở đâu trong quy trình khoa học dữ liệu (xem hình 6.18).



**Hình 6.18** Quy trình khoa học dữ liệu bước 2: truy xuất dữ liệu. Trong trường hợp này không có dữ liệu nội bộ; tất cả dữ liệu sẽ được lấy từ Wikipedia.

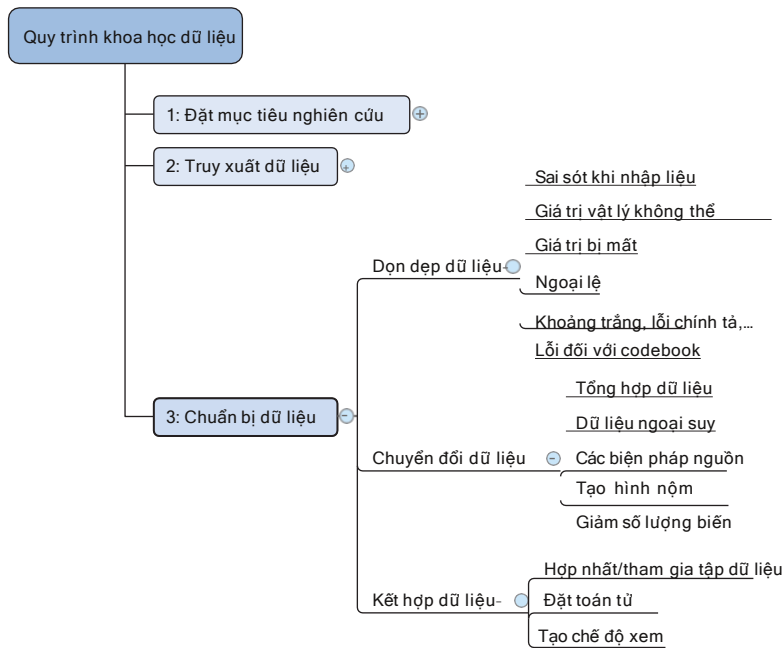
Như thể hiện trong hình 6.18, bạn có thể có hai nguồn: dữ liệu nội bộ và dữ liệu bên ngoài.

- *Internal data - dữ liệu nội bộ*—Bạn không có thông tin về bệnh nằm xung quanh. Nếu bạn hiện đang làm việc cho một công ty dược phẩm hoặc bệnh viện, bạn có thể may mắn hơn.
- *External data - dữ liệu bên ngoài*—Tất cả những gì bạn có thể sử dụng cho trường hợp này là dữ liệu bên ngoài. Bạn có một số lựa chọn, nhưng bạn sẽ chọn Wikipedia.

Khi bạn lấy dữ liệu từ Wikipedia, bạn sẽ cần lưu trữ nó trong chỉ mục Elasticsearch cục bộ của mình, nhưng trước khi làm điều đó, bạn cần chuẩn bị dữ liệu. Khi dữ liệu đã được nhập vào chỉ mục Elasticsearch, nó không thể thay đổi được; tất cả những gì bạn có thể làm sau đó là truy vấn nó. Xem tổng quan về chuẩn bị dữ liệu trong hình 6.19.

Như thể hiện trong hình 6.19, có ba loại chuẩn bị dữ liệu riêng biệt cần xem xét:

- *Data cleansing - dọn dẹp dữ liệu*—Dữ liệu bạn lấy từ Wikipedia có thể không đầy đủ hoặc sai sót. Có thể xảy ra lỗi nhập dữ liệu và lỗi chính tả—không loại trừ cả thông tin sai lệch. May mắn thay, bạn không cần danh sách đầy đủ các bệnh và bạn có thể xử lý các lỗi chính tả khi tìm kiếm; nhiều hơn về điều đó sau này. Nhờ thư viện Wikipedia Python, dữ liệu văn bản bạn sẽ nhận được đã khá rõ ràng. Nếu bạn xóa nó theo cách thủ công, bạn cần thêm tính năng làm sạch HTML, xóa tất cả các thẻ HTML. Sự thật của vấn đề là tìm kiếm toàn văn bản có xu hướng khá mạnh mẽ đối với các lỗi phổ biến như giá trị không chính xác. Ngay cả khi bạn cố tình bỏ vào các thẻ HTML, chúng sẽ không thể ảnh hưởng đến kết quả; các thẻ HTML quá khác so với ngôn ngữ bình thường để can thiệp.



Hình 6.19 Quy trình khoa học dữ liệu bước 3: chuẩn bị dữ liệu

- *Data transformation - chuyển đổi dữ liệu*—Bạn không cần phải chuyển đổi dữ liệu nhiều vào thời điểm này; bạn muốn tìm kiếm nó. Nhưng bạn sẽ phân biệt giữa tiêu đề trang, tên bệnh và nội dung trang. Sự khác biệt này gần như là bắt buộc đối với việc diễn giải kết quả tìm kiếm.
- *Combining data - kết hợp dữ liệu*—Tất cả dữ liệu được lấy từ một nguồn duy nhất trong trường hợp này, vì vậy bạn thực sự không cần phải kết hợp dữ liệu. Một phần mở rộng khả thi cho bài tập này là lấy dữ liệu về bệnh tật từ một nguồn khác và so khớp các bệnh tật. Đây không phải là nhiệm vụ tầm thường vì không có mã định danh duy nhất và tên thường hơi khác nhau.

Bạn chỉ có thể làm sạch dữ liệu ở hai giai đoạn: khi sử dụng chương trình Python kết nối Wikipedia với Elasticsearch và khi chạy hệ thống lập chỉ mục nội bộ của Elasticsearch:

- *Python*—Tại đây, bạn xác định dữ liệu nào bạn sẽ cho phép lưu trữ bởi kho lưu trữ tài liệu của mình, nhưng bạn sẽ không xóa dữ liệu hoặc chuyển đổi dữ liệu ở giai đoạn này, bởi vì Elasticsearch làm tốt hơn với ít nỗ lực hơn.
- *Elasticsearch*—Elasticsearch sẽ xử lý thao tác dữ liệu (tạo chỉ mục). Bạn vẫn có thể tác động đến quá trình này và bạn sẽ làm điều đó một cách rõ ràng hơn ở phần sau của chương này.

Bây giờ bạn đã có cái nhìn tổng quan về các bước sắp tới, hãy bắt tay vào việc. Nếu bạn đã làm theo các hướng dẫn trong phụ lục, thì bây giờ bạn sẽ có một phiên bản Elasticsearch cục bộ được thiết lập và chạy. Đầu tiên là truy xuất dữ liệu: bạn cần thông tin về các bệnh khác nhau. Bạn có một số cách để có được loại dữ liệu đó. Bạn có thể yêu cầu các công ty cung cấp dữ liệu của họ hoặc lấy dữ liệu từ Freebase hoặc các nguồn dữ liệu mở và miễn phí khác. Thu thập dữ liệu của bạn có thể là một thách thức, nhưng với ví dụ này, bạn sẽ lấy dữ liệu đó từ Wikipedia. Điều này hơi kì cục vì các tìm kiếm trên trang web Wikipedia được xử lý bởi Elasticsearch. Wikipedia đã từng xây dựng hệ thống của riêng mình dựa trên Apache Lucene, nhưng hệ thống này trở nên không thể bảo trì được và kể từ tháng 1 năm 2014, Wikipedia đã bắt đầu sử dụng Elasticsearch để thay thế.

Wikipedia có một trang Danh sách các bệnh, như thể hiện trong hình 6.20. Từ đây bạn có thể mượn dữ liệu từ danh sách theo thứ tự bảng chữ cái.

## 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](#)

### Diseases

Alphabetical list

0-9 · [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](#)

See also

[Health](#) · [Exercise](#) · [Nutrition](#)

V · T · E

Hình 6.20 Trang Danh sách bệnh của Wikipedia, điểm khởi đầu cho việc truy xuất dữ liệu của bạn

Bạn biết bạn muốn dữ liệu nào; bây giờ hãy đi lấy nó. Bạn có thể tải xuống toàn bộ kết xuất dữ liệu Wikipedia. Nếu muốn, bạn có thể tải xuống từ [http://meta.wikimedia.org/wiki/Data\\_dump\\_torrents#enwiki](http://meta.wikimedia.org/wiki/Data_dump_torrents#enwiki).

Tất nhiên, nếu bạn lập chỉ mục cho toàn bộ Wikipedia, thì việc lập chỉ mục sẽ yêu cầu khoảng 40 GB dung lượng lưu trữ. Hãy thoải mái sử dụng giải pháp này, nhưng vì mục đích duy trì dung lượng lưu trữ và băng thông, trong cuốn sách này, chúng tôi sẽ giới hạn chỉ lấy dữ liệu mà chúng tôi dự định sử dụng. Một tùy chọn khác là cào các trang bạn yêu cầu. Giống như Google, bạn có thể làm cho chương trình thu thập thông tin qua các trang và truy xuất toàn bộ HTML được hiển thị. Điều này sẽ thực hiện được thủ thuật, nhưng cuối cùng bạn sẽ nhận được HTML thực, vì vậy bạn cần dọn sạch nó trước khi lập chỉ mục cho nó. Ngoài ra, trừ khi bạn là Google, các trang web không thích các trình thu thập dữ liệu tìm kiếm các trang web của họ. Điều này tạo ra một lượng lưu lượng truy cập cao không cần thiết và nếu có đủ người gửi trình thu thập thông tin, nó có thể làm máy chủ HTTP quá tải và làm mất niềm vui của mọi

người. Gửi hàng tỷ yêu cầu cùng lúc cũng là một trong những cách tấn công từ chối dịch vụ - Denial Of Service (DoA) được thực hiện. Nếu bạn cần cào một trang web, hãy viết kịch bản trong khoảng thời gian giữa mỗi yêu cầu trang. Bằng cách này, trình quét của bạn mô phỏng chặt chẽ hơn hành vi của một khách truy cập trang web thông thường và bạn sẽ không làm nổ tung máy chủ của họ.

May mắn thay, những người tạo ra Wikipedia đủ thông minh để biết rằng đây chính xác là điều sẽ xảy ra với tất cả những thông tin này được mở cho tất cả mọi người. Họ đã đặt một API để bạn có thể lấy thông tin của mình một cách an toàn. Bạn có thể đọc thêm về nó tại [http://www.mediawiki.org/wiki/API:Main\\_page](http://www.mediawiki.org/wiki/API:Main_page).

Bạn sẽ rút ra từ API. Và Python sẽ không phải là Python nếu nó chưa có thư viện để thực hiện công việc. Thực tế có một số, nhưng cách dễ nhất sẽ đủ cho nhu cầu của bạn: Wikipedia.

Kích hoạt môi trường ảo Python của bạn và cài đặt tất cả các thư viện bạn cần cho phần còn lại của cuốn sách:

```
pip install wikipedia
pip install Elasticsearch
```

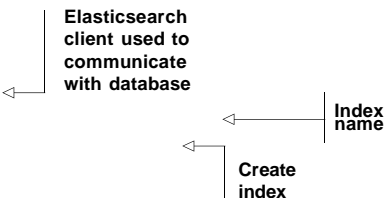
Bạn sẽ sử dụng Wikipedia để khai thác Wikipedia. Elasticsearch là thư viện chính của Elasticsearch Python; với nó, bạn có thể giao tiếp với cơ sở dữ liệu của mình.

Mở trình thông dịch Python yêu thích của bạn và nhập các thư viện cần thiết:

```
from elasticsearch import Elasticsearch
import wikipedia
```

Bạn sẽ lấy dữ liệu từ API Wikipedia và đồng thời lập chỉ mục trên phiên bản Elasticsearch cục bộ của bạn, vì vậy trước tiên bạn cần chuẩn bị nó để chấp nhận dữ liệu.

```
client = Elasticsearch()
indexName = "medical"
client.indices.create(index=indexName)
```



Điều đầu tiên bạn cần là một khách hàng. `Elasticsearch()` có thể được khởi tạo bằng một địa chỉ nhưng mặc định là `localhost:9200`. `Elasticsearch()` và `Elasticsearch('localhost:9200')` là giống nhau: ứng dụng khách của bạn được kết nối với nút Elasticsearch cục bộ của bạn. Sau đó, bạn tạo một chỉ mục có tên "medical". Nếu mọi việc suôn sẻ, bạn sẽ thấy một biểu tượng trả lời "acknowledge:true", như thể hiện trong hình 6.21.

Elasticsearch tuyên bố là không có lược đồ, nghĩa là bạn có thể sử dụng Elasticsearch mà không cần xác định lược đồ cơ sở dữ liệu và không cho Elasticsearch biết loại dữ liệu mong

```
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}
```

Hình 6.21 Tạo chỉ mục Elasticsearch bằng Python-Elasticsearch

đợi. Mặc dù điều này đúng với các trường hợp đơn giản, nhưng về lâu dài, bạn không thể tránh khỏi việc có một lược đồ, vì vậy, hãy tạo một lược đồ, như minh họa trong danh sách sau đây.

#### Listing 6.1 Thêm ánh xạ vào loại tài liệu

```
diseaseMapping = {
    'properties': {
        'name': {'type': 'string'},
        'title': {'type': 'string'},
        'fulltext': {'type': 'string'}
    }
}
client.indices.put_mapping(index=indexName,
doc_type='diseases',body=diseaseMapping)
```

← Định nghĩa một ánh xạ và quy nó cho loại tài liệu bệnh.

← Loại tài liệu "disease" được cập nhật bằng bản đồ. Bây giờ chúng ta xác định dữ liệu mà nó mong đợi.

Bằng cách này, bạn nói với Elasticsearch rằng chỉ mục của bạn sẽ có một loại tài liệu được gọi là "disease", và bạn cung cấp cho nó loại trường cho từng trường. Bạn có ba trường trong một tài liệu về bệnh: name, title, và fulltext, tất cả đều thuộc loại string. Nếu bạn không cung cấp bản đồ, Elasticsearch sẽ đoán loại của chúng bằng cách xem mục nhập đầu tiên mà nó nhận được. Nếu nó không nhận ra trường là boolean, double, float, long, integer, hoặc date, nó sẽ đặt nó thành string. Trong trường hợp này, bạn không cần chỉ định ánh xạ theo cách thủ công.

Bây giờ hãy chuyển sang Wikipedia. Điều đầu tiên bạn muốn làm là tìm nạp trang Danh sách bệnh, bởi vì đây là điểm bắt đầu của bạn để khám phá thêm:

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

Bây giờ bạn đã có trang đầu tiên của mình, nhưng bạn quan tâm nhiều hơn đến các trang liệt kê vì chúng chứa liên kết đến các bệnh. Kiểm tra các liên kết:

```
dl.links
```

Trang Danh sách bệnh đi kèm với nhiều liên kết hơn bạn sẽ sử dụng. Hình 6.22 hiển thị các danh sách theo thứ tự chữ cái bắt đầu từ liên kết thứ mười sáu.

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

```
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)']
```

**Hình 6.22** Các liên kết trên trang Wikipedia Danh sách các bệnh. Nó có nhiều liên kết hơn bạn cần.

Trang này có một loạt các liên kết đáng kể, nhưng bạn chỉ quan tâm đến danh sách theo thứ tự chữ cái, vì vậy chỉ giữ lại những danh sách sau:

```
diseaseListArray = []
for link in dl.links[15:42]:
    try:
        diseaseListArray.append(wikipedia.page(link))
    except Exception,e:
        print str(e)
```

Bạn có thể nhận thấy rằng tập hợp con được mã hóa cứng, bởi vì bạn biết chúng là các mục từ thứ 16 đến 43 trong mảng. Nếu Wikipedia thêm dù chỉ một liên kết trước những liên kết mà bạn quan tâm, nó sẽ làm hỏng kết quả. Một cách thực hành tốt hơn là sử dụng các biểu thức chính quy cho tác vụ này. Đối với mục đích khám phá, mã hóa cứng các số mục nhập là tốt, nhưng nếu biểu thức chính quy là bản chất thứ hai đối với bạn hoặc bạn có ý định biến mã này thành một công việc hàng loạt, thì nên sử dụng biểu thức chính quy. Bạn có thể tìm thêm thông tin về chúng tại <https://docs.python.org/2/howto/regex.html>.

Một khả năng cho phiên bản regex sẽ là đoạn mã sau.

```
diseaseListArray = []
check = re.compile("List of diseases*")
for link in dl.links:
    if check.match(link):
        try:
            diseaseListArray.append(wikipedia.page(link))
        except Exception,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)'>]
```

Hình 6.23 Danh sách bệnh đầu tiên Wikipedia, “list of diseases (0-9)”

Hình 6.23 cho thấy mục đầu tiên của những gì bạn đang theo đuổi: bản thân các bệnh tật.

diseaseListArray[0].links

Đã đến lúc lập chỉ mục các bệnh. Sau khi chúng được lập chỉ mục, cả việc nhập dữ liệu và chuẩn bị dữ liệu đều kết thúc một cách hiệu quả, như thể hiện trong danh sách sau.

### Listing 6.2 Lập chỉ mục các bệnh từ Wikipedia

```
checkList = [["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 = 'diseases'
for diseaselistNumber, diseaselist in enumerate(diseaseListArray):
    for disease in diseaselist.links:
        try:
            if disease[0] in checklist[diseaselistNumber]
            and disease[0:3] != "List":
                currentPage = wikipedia.page(disease)
                client.index(index=indexName,
                             doc_type=docType,id = disease, body={"name": disease,
                             "title":currentPage.title,
                             "fulltext":currentPage.content})
            except Exception,e:
                print str(e)
```

Vòng lặp danh sách các bệnh

Danh sách kiểm tra là một mảng chứa một mảng các ký tự đầu tiên hợp lệ. Nếu một căn bệnh không tuân thủ, bỏ qua nó.

Loại tài liệu bạn lấy chỉ mục

Vòng lặp qua danh sách các liên kết cho các bệnh.

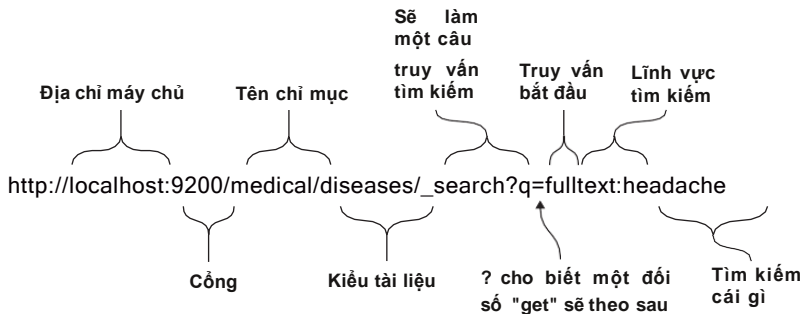
Trước tiên kiểm tra xem đó có phải là bệnh không, sau đó lập chỉ mục cho nó.

Vì mỗi trang trong danh sách sẽ có các liên kết bạn không cần, hãy kiểm tra xem mục nhập có phải là bệnh hay không. Bạn chỉ ra cho mỗi danh sách ký tự mà bệnh bắt đầu, vì vậy hãy kiểm tra điều này. Ngoài ra, bạn loại trừ các liên kết bắt đầu bằng “danh sách” vì những liên kết này sẽ bật lên khi bạn đến danh sách L các bệnh. Việc kiểm tra khá ngây thơ, nhưng chi phí để có một vài mục nhập không mong muốn là khá thấp vì các thuật toán tìm kiếm sẽ loại trừ các kết quả không liên quan khi bạn bắt đầu truy vấn. Đối với mỗi bệnh, bạn lập chỉ mục tên bệnh và toàn văn của trang. Tên này cũng được sử dụng làm ID chỉ mục của nó; cái này hữu ích cho một số



tính năng Elasticsearch nâng cao mà còn để tra cứu nhanh trong trình duyệt. Ví dụ: hãy thử URL này trong trình duyệt của bạn: `http://localhost:9200/medical/diseases/11%20beta%20hydroxylase%20deficiency`. Tiêu đề được lập chỉ mục riêng; trong hầu hết các trường hợp, tên liên kết và tiêu đề trang sẽ giống hệt nhau và đôi khi tiêu đề sẽ chứa một tên thay thế cho bệnh.

Với ít nhất một số bệnh được lập chỉ mục, có thể sử dụng URI Elasticsearch để tra cứu đơn giản. Hãy xem một tìm kiếm đầy đủ cho từ *headache* - *đau đầu* trong hình 6.24. Bạn đã có thể làm điều này trong khi lập chỉ mục; Elasticsearch có thể cập nhật một chỉ mục và trả về các truy vấn cho nó cùng một lúc.



Hình 6.24 Ví dụ xây dựng URL Elasticsearch

Nếu bạn không truy vấn chỉ mục, bạn vẫn có thể nhận được một vài kết quả mà không cần biết gì về chỉ mục. Chỉ định `http://localhost:9200/Medical/diseases/_search` sẽ trả về năm kết quả đầu tiên. Để có chế độ xem dữ liệu có cấu trúc hơn, bạn có thể yêu cầu ánh xạ của loại tài liệu này tại `http://localhost:9200/medical/disease/_mapping?pretty`. Đối số `get` hiển thị kết quả JSON được trả về ở định dạng dễ đọc hơn, như có thể thấy trong hình 6.25. Ánh xạ có vẻ giống như cách bạn đã chỉ định: tất cả các trường đều là loại `string`.

URL Elasticsearch chắc chắn hữu ích, nhưng nó sẽ không đủ cho nhu cầu của bạn. Bạn vẫn còn bệnh cần chẩn đoán và để làm điều này, bạn sẽ gửi các yêu cầu POST tới Elasticsearch thông qua thư viện Elasticsearch Python.

Sau khi hoàn thành việc truy xuất và chuẩn bị dữ liệu, bạn có thể chuyển sang khai thác dữ liệu của mình.

```

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

```

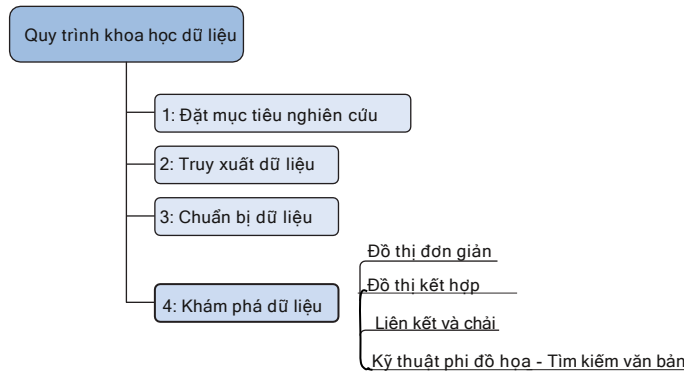
Hình 6.25 Loại tài liệu bệnh ánh xạ qua URL Elasticsearch

### 6.2.3 Bước 4: Khai thác dữ liệu

Nó không phải là bệnh lupus. Nó không bao giờ là bệnh lupus!

— Tiến sĩ House trong phim House M.D

Khai thác dữ liệu là điều đánh dấu nghiên cứu điển hình này, bởi vì mục tiêu chính của dự án (chẩn đoán bệnh) là một cách cụ thể để khám phá dữ liệu bằng cách truy vấn các triệu chứng bệnh. Hình 6.26 trình bày một số kỹ thuật khám phá dữ liệu, nhưng trong trường hợp này, kỹ thuật này không mang tính đồ họa: diễn giải các kết quả truy vấn tìm kiếm văn bản.



Hình 6.26 Quy trình khoa học dữ liệu bước 4: khám phá dữ liệu

Thời điểm của sự thật là đây: bạn có thể tìm ra một số bệnh bằng cách cung cấp cho công cụ tìm kiếm của bạn các triệu chứng của chúng không? Trước tiên, hãy đảm bảo rằng bạn có những điều cơ bản để thiết lập và chạy. Nhập thư viện Elasticsearch và xác định cài đặt tìm kiếm toàn cầu:

```

from elasticsearch import Elasticsearch
client = Elasticsearch()
indexName = "medical"
docType="diseases"
searchFrom = 0
searchSize= 3

```

Bạn sẽ chỉ trả về ba kết quả đầu tiên; mặc định là năm.

Elasticsearch có ngôn ngữ truy vấn JSON phức tạp; mọi tìm kiếm là một yêu cầu POST tới máy chủ và sẽ được trả lời bằng câu trả lời JSON. Đại khái, ngôn ngữ bao gồm ba phần lớn: truy vấn, bộ lọc và tổng hợp. Một *query* - *truy vấn* lấy các từ khóa tìm kiếm và đưa chúng qua một hoặc nhiều bộ phân tích trước khi các từ được tra cứu trong chỉ mục. Chúng ta sẽ tìm hiểu sâu hơn về máy phân tích ở phần sau của chương này. Một *filter* - *lọc* lấy các từ khóa giống như một truy vấn nhưng không cố gắng phân tích những gì bạn cung cấp cho nó; nó lọc theo các điều kiện mà ta cung cấp. Do đó, các bộ lọc ít phức tạp hơn nhưng hiệu quả hơn nhiều lần vì

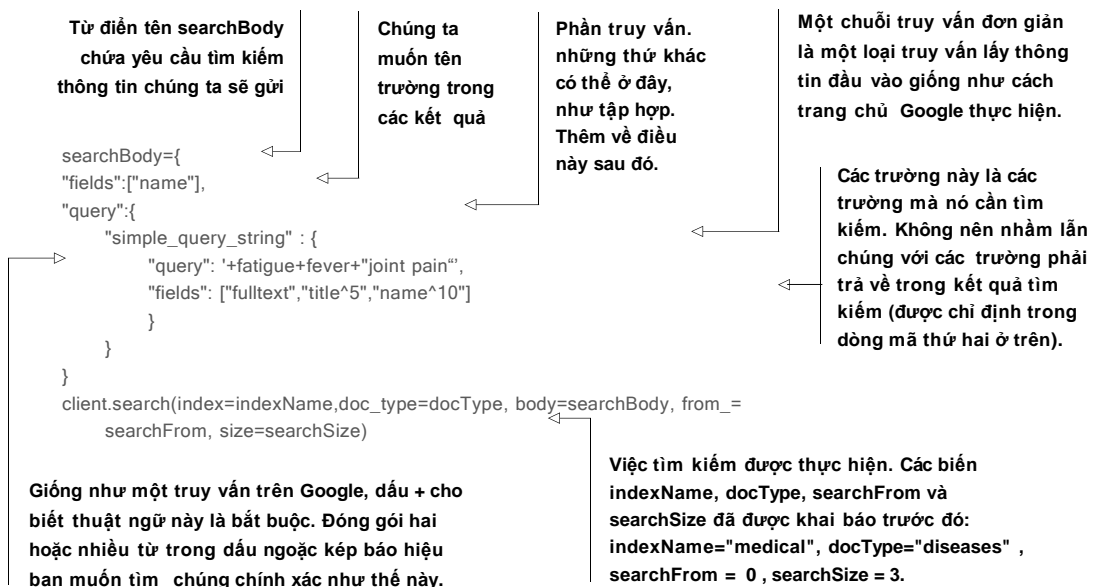
chúng cũng được lưu trữ tạm thời trong Elasticsearch trong trường hợp bạn sử dụng cùng một bộ lọc hai lần. *Aggregations* - *tổng hợp* có thể được so sánh với nhóm SQL; nhóm các từ sẽ được tạo và đối với mỗi nhóm số liệu thống kê có liên quan có thể được tính toán. Mỗi bộ phận trong số ba bộ phận này có vô số tùy chọn và tính năng, khiến việc xây dựng toàn bộ ngôn ngữ ở đây là không thể. May mắn thay, không cần phải đi sâu vào sự phức tạp mà các truy vấn Elasticsearch có thể biểu thị. Chúng ta sẽ sử dụng “Query string query language” - “Ngôn ngữ truy vấn chuỗi truy vấn”, một cách để truy vấn dữ liệu gần giống với ngôn ngữ truy vấn tìm kiếm của Google. Ví dụ: nếu bạn muốn cụm từ tìm kiếm là bắt buộc, bạn thêm dấu cộng (+); nếu bạn muốn loại trừ cụm từ tìm kiếm, bạn sử dụng dấu trừ (-). Truy vấn Elasticsearch không được khuyến nghị vì nó làm giảm hiệu suất; trước tiên, công cụ tìm kiếm cần dịch chuỗi truy vấn sang ngôn ngữ truy vấn JSON gốc của nó. Nhưng với mục đích của bạn, nó sẽ hoạt động tốt; hơn nữa, hiệu suất sẽ không phải là một yếu tố trong vài nghìn bản ghi mà bạn có trong chỉ mục của mình. Bây giờ là lúc để truy vấn dữ liệu bệnh tật của bạn.

### MỤC TIÊU CHÍNH CỦA DỰ ÁN: CHẨN ĐOÁN BỆNH BẰNG TRIỆU CHỨNG CỦA BỆNH

Nếu bạn đã từng xem bộ phim truyền hình nổi tiếng *House M.D.*, câu “Không bao giờ là bệnh lupus” nghe có vẻ quen thuộc. Lupus là một loại bệnh tự miễn dịch, trong đó hệ thống miễn dịch của cơ thể tấn công các bộ phận khỏe mạnh của cơ thể. Hãy xem những triệu chứng mà công cụ tìm kiếm của bạn sẽ cần để xác định rằng bạn đang tìm kiếm bệnh lupus.

Bắt đầu với ba triệu chứng: fatigue - mệt mỏi, fever - sốt và joint pain - đau khớp. Bệnh nhân tương tượng của bạn có tất cả ba người trong số họ (và hơn thế nữa), vì vậy hãy bắt buộc phải có tất cả bằng cách thêm dấu cộng trước mỗi người:

#### Listing 6.3 Truy vấn Elasticsearch “chuỗi truy vấn đơn giản” với ba từ khóa bắt buộc



Trong phần `searchBody`, có cấu trúc JSON, bạn chỉ định các trường bạn muốn thấy được trả về, trong trường hợp này, tên của bệnh là đủ. Bạn sử dụng cú pháp chuỗi truy vấn để tìm kiếm trong tất cả các trường được lập chỉ mục: `fulltext`, `title`, và `name`. Bằng cách thêm `^` bạn có thể gán trọng số cho mỗi trường. Nếu một triệu chứng xuất hiện trong tiêu đề, thì nó quan trọng gấp năm lần so với trong văn bản mở; nếu nó xuất hiện trong chính cái tên, thì nó được coi là quan trọng gấp mười lần. Lưu ý cách cụm từ "joint pain" được đặt trong một cặp dấu ngoặc kép. Nếu bạn không có dấu `"", joint và pain sẽ được coi là hai từ khóa riêng biệt chứ không phải là một cụm từ. Trong Elasticsearch, cái này được gọi là phrase matching. Hãy xem kết quả trong hình 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}
```

Lupus không trong top 3 bệnh tật được kết quả trả về.

34 bệnh được tìm thấy

Hình 6.27 Tìm kiếm Lupus đầu tiên với 34 kết quả

Hình 6.27 cho thấy ba kết quả hàng đầu được trả về trong số 34 bệnh phù hợp. Các kết quả được sắp xếp theo điểm phù hợp của chúng, biến `_score`. Điểm phù hợp không phải là điều đơn giản để giải thích; nó sẽ xem xét mức độ phù hợp của căn bệnh với truy vấn của bạn và số lần một từ khóa được tìm thấy, trọng số bạn đưa ra, v.v. Hiện tại, bệnh lupus thậm chí không xuất hiện trong ba kết quả hàng đầu. May mắn cho bạn, bệnh lupus có một triệu chứng khác biệt: a rash - phát ban. Phát ban không phải lúc nào cũng xuất hiện trên khuôn mặt của một người, nhưng nó vẫn xảy ra và đây là lý do khiến bệnh lupus có tên: phát ban trên mặt khiến mọi người trông giống như một con sói. Bệnh nhân của bạn bị phát ban nhưng không phải là phát ban đặc trưng trên mặt, vì vậy hãy thêm "rash" – "phát ban" vào các triệu chứng mà không đề cập đến khuôn mặt.

"query": '+fatigue+fever+"joint pain"+rash',

```
{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}
```

**Hình 6.28** Lần tìm kiếm thứ hai về bệnh Lupus với sáu kết quả và bệnh Lupus nằm trong ba kết quả hàng đầu

Kết quả tìm kiếm mới được thể hiện trong hình 6.28.

Giờ đây, kết quả đã được thu hẹp xuống còn sáu và bệnh lupus nằm trong top ba. Tại thời điểm này, công cụ tìm kiếm cho biết *Human Granulocytic Ehrlichiosis - Ehrlichiosis bạch cầu hạt ở người* (HGE) có nhiều khả năng hơn. HGE là một căn bệnh lây lan qua bọ ve, giống như bệnh Lyme khét tiếng. Đến bây giờ, một bác sĩ có năng lực sẽ tìm ra căn bệnh nào đang gây ra cho bệnh nhân của bạn, bởi vì khi xác định bệnh có nhiều yếu tố tác động, nhiều hơn những gì bạn có thể cung cấp cho công cụ tìm kiếm khiêm tốn của mình. Chẳng hạn, phát ban chỉ xảy ra ở 10% HGE và 50% bệnh nhân lupus. Lupus xuất hiện từ từ, trong khi HGE được kích hoạt bởi vết cắn của ve. Cơ sở dữ liệu máy học tiên tiến được cung cấp tất cả thông tin này theo cách có cấu trúc hơn có thể đưa ra chẩn đoán với độ chắc chắn cao hơn nhiều. Cho rằng bạn cần phải làm gì với các trang Wikipedia, bạn cần một triệu chứng khác để xác nhận rằng đó là bệnh lupus. Bệnh nhân bị chest pain - đau ngực, vì vậy hãy thêm điều này vào danh sách.

```
"query": "+fatigue+fever+"joint pain"+rash+"chest pain",
```

Kết quả được thể hiện trong hình 6.29.

Có vẻ như đó là bệnh lupus. Phải mất một thời gian để đi đến kết luận này, nhưng bạn đã đạt được điều đó. Tất nhiên, bạn đã bị giới hạn trong cách trình bày các triệu chứng của Elasticsearch. Bạn chỉ sử dụng các thuật ngữ đơn lẻ (“fatigue” - “mệt mỏi”) hoặc các cụm từ theo nghĩa đen (“joint pain” - “đau khớp”). Điều này phù hợp với ví dụ này, nhưng Elasticsearch linh hoạt hơn thế này. Nó có thể nhận các biểu thức chính quy và thực hiện tìm kiếm mờ, nhưng điều đó nằm ngoài phạm vi của cuốn sách này, mặc dù một số ví dụ được bao gồm trong mã có thể tải xuống.

```
{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}
```

Hình 6.29 Tìm kiếm  
Lupus lần thứ ba: có đủ  
triệu chứng để xác định  
nó phải là bệnh lupus

## XỬ LÝ LỖI CHÍNH TẢ: DAMERAU-LEVENSHTEIN

Giả sử ai đó đã gõ “lupsu” thay vì “lupus”. Lỗi chính tả xảy ra mọi lúc và trong tất cả các loại tài liệu do con người tạo ra. Đề đối phó với dữ liệu này, các nhà khoa học thường sử dụng Damerau-Levenshtein. Khoảng cách Damerau-Levenshtein giữa hai chuỗi là số thao tác cần thiết để biến chuỗi này thành chuỗi kia. Bốn thao tác cần để tính khoảng cách:

- *Deletion* - xóa—Xóa một ký tự khỏi chuỗi.
- *Insertion* - chèn—Thêm một ký tự vào chuỗi.
- *Substitution* - thay thế—Thay thế một ký tự cho một ký tự khác. Nếu không thay thế được tính là một thao tác, việc thay đổi ký tự này thành ký tự khác sẽ thực hiện hai thao tác: một lần xóa và một lần chèn.
- *Transposition of two adjacent characters* - chuyển vị của hai ký tự liền kề—Hoán đổi hai ký tự liền kề.

Thao tác cuối cùng (transposition - chuyển vị) là điều tạo nên sự khác biệt giữa khoảng cách Levenshtein truyền thống và khoảng cách Damerau-Levenshtein. Chính thao tác cuối cùng này đã làm cho lỗi chính tả mắc chứng khó đọc của chúng ta nằm trong giới hạn chấp nhận được. Damerau-Levenshtein bỏ qua những lỗi chuyển vị này, điều này làm cho nó trở nên tuyệt vời cho các công cụ tìm kiếm, nhưng nó cũng được sử dụng cho những thứ khác như tính toán sự khác biệt giữa các chuỗi DNA.

Hình 6.30 cho thấy cách chuyển từ “lupsu” thành “lupus” được thực hiện với một lần chuyển vị.

Lupsu  $\longrightarrow$  Lupsu  $\longrightarrow$  Lupus

Hình 6.30 Chuyển vị ký tự liền kề là một trong các thao tác trong khoảng cách Damerau-Levenshtein. Ba cái còn lại là chèn, xóa và thay thế.

Chỉ với điều này, bạn đã đạt được mục tiêu đầu tiên của mình: *chẩn đoán bệnh*. Nhưng đừng quên mục tiêu dự án phụ của bạn: *lập hồ sơ bệnh tật*.

## MỤC TIÊU PHỤ CỦA DỰ ÁN: LẬP HỒ SƠ BỆNH

Những gì bạn muốn là một danh sách các từ khóa phù hợp với căn bệnh bạn đã chọn. Đối với điều này, bạn sẽ sử dụng tập hợp các thuật ngữ quan trọng. Cách tính điểm để xác định từ nào có ý nghĩa một lần nữa là sự kết hợp của nhiều yếu tố, nhưng về cơ bản nó chỉ là một phép so

sánh về số lần một thuật ngữ được tìm thấy trong tập kết quả so với tất cả các tài liệu khác. Bằng cách này, Elasticsearch cấu hình tập kết quả của bạn bằng cách cung cấp các từ khóa phân biệt nó với các dữ liệu khác. Hãy làm điều đó với diabetes - bệnh tiểu đường, một căn bệnh phổ biến có thể ở nhiều dạng:

#### Listing 6.4 Các thuật ngữ quan trọng Truy vấn Elasticsearch cho "tiểu đường"



Bạn thấy mã mới ở đây. Bạn đã loại bỏ tìm kiếm chuỗi truy vấn và thay vào đó sử dụng bộ lọc. Bộ lọc được gói gọn trong phần truy vấn vì các truy vấn tìm kiếm có thể được kết hợp với các bộ lọc. Nó không xảy ra trong ví dụ này, nhưng khi điều này xảy ra, Elasticsearch trước tiên sẽ áp dụng bộ lọc, nó hiệu quả hơn nhiều trước khi thử tìm kiếm. Nếu bạn biết mình muốn tìm kiếm trong một tập hợp con dữ liệu của mình, bạn nên thêm một bộ lọc để tạo tập hợp con này trước tiên. Để chứng minh điều này, hãy xem xét hai đoạn mã sau. Chúng mang lại kết quả giống nhau nhưng chúng không hoàn toàn giống nhau.



Một chuỗi truy vấn đơn giản tìm kiếm “diabetes” – “bệnh tiểu đường” trong tên bệnh:

```
"query":{
  "simple_query_string": {
    "query": 'diabetes',
    "fields": ["name"]
  }
}
```

Một thuật ngữ lọc lọc trong tất cả các bệnh có tên “diabetes” - “tiểu đường”:

```
"query":{
  "filtered": {
    "filter": {
      "term": {'name':'diabetes'}
    }
  }
}
```

Mặc dù nó sẽ không hiển thị trên lượng dữ liệu nhỏ mà bạn tùy ý sử dụng, nhưng bộ lọc nhanh hơn nhiều so với tìm kiếm. Truy vấn tìm kiếm sẽ tính điểm tìm kiếm cho từng bệnh và xếp hạng chúng tương ứng, trong khi bộ lọc chỉ lọc ra tất cả những bệnh không tuân thủ. Do đó, một bộ lọc ít phức tạp hơn nhiều so với một tìm kiếm thực tế: đó là “có” hoặc “không” và điều này thể hiện rõ ở đầu ra. Điểm số là 1 cho mọi thứ; không có sự phân biệt nào được thực hiện trong tập kết quả. Đầu ra hiện tại bao gồm hai phần do tập hợp các thuật ngữ quan trọng. Trước đây bạn chỉ có số lần truy cập; bây giờ bạn có lượt truy cập và tổng hợp. Đầu tiên, hãy xem các lần truy cập trong hình 6.31.

Điều này bây giờ trông có vẻ quen thuộc với một ngoại lệ đáng chú ý: tất cả các kết quả đều có điểm là 1. Ngoài việc dễ thực hiện hơn, một bộ lọc được Elasticsearch lưu vào bộ đệm cho

```
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}
```

Hình 6.31 Lượt truy cập đầu ra của truy vấn đã lọc với bộ lọc “diabetes” trên tên bệnh



một lúc. Bằng cách này, các yêu cầu tiếp theo với cùng một bộ lọc thậm chí còn nhanh hơn, dẫn đến lợi thế về hiệu suất rất lớn so với các truy vấn tìm kiếm.

Khi nào bạn nên sử dụng bộ lọc và khi nào truy vấn tìm kiếm? Quy tắc rất đơn giản: sử dụng bộ lọc bất cứ khi nào có thể và sử dụng truy vấn tìm kiếm để tìm kiếm toàn văn khi cần xếp hạng giữa các kết quả để nhận được kết quả thú vị nhất ở trên cùng.

Bây giờ hãy xem các số hạng có ý nghĩa trong hình 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},
```

**Hình 6.32** Tổng hợp các thuật ngữ có ý nghĩa về bệnh tiểu đường, năm từ khóa đầu tiên

Nếu bạn nhìn vào năm từ khóa đầu tiên trong hình 6.32, bạn sẽ thấy rằng bốn từ khóa hàng đầu có liên quan đến nguồn gốc của bệnh tiểu đường. Đoạn Wikipedia sau đây cung cấp trợ giúp:

Từ tiểu đường diabetes (/ ˌdaɪ.əbiː tiː z/ hoặc / ˌdaɪ.əˈʃʊtɪs/) xuất phát từ tiếng Latin *diabe-te-s*, lại xuất phát từ tiếng Hy Lạp cổ đại có nghĩa đen là “người qua đường; một xi phông” [69]. Bác sĩ Hy Lạp cổ đại Aretaeus ở Cappadocia (fl. Thế kỷ 1 CN) đã sử dụng từ đó, với ý nghĩa dự kiến là “tiểu ra quá nhiều nước tiểu,” làm tên cho căn bệnh [70, 71, 72]. Cuối cùng, từ này bắt nguồn từ tiếng Hy Lạp (*diabainein*), nghĩa là “đi qua”, [69] bao gồm (*dia-*), nghĩa là “đi qua” và (*bainein*), nghĩa là “đi” [70]. Từ “diabetes” lần đầu tiên được ghi lại bằng tiếng Anh, dưới dạng bệnh tiểu đường, trong một văn bản y tế được viết vào khoảng năm 1425.

—Trang *Diabetes\_mellitus* của Wikipedia

Điều này cho bạn biết từ *diabetes* đến từ: “một người qua đường; một siphon” trong tiếng Hy Lạp. Nó cũng đề cập đến *diabainein* và *bainein*. Bạn có thể đã biết rằng hầu hết các từ khóa

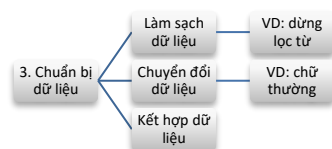
có liên quan cho một căn bệnh sẽ là định nghĩa và nguồn gốc thực tế. May mắn thay, chúng tôi đã yêu cầu 30 từ khóa, vì vậy hãy chọn một số từ khóa thú vị hơn chẳng hạn như *ndi*. *ndi* là phiên bản chữ thường của *NDI*, hay còn gọi là “Nephrogenic Diabetes Insipidus” - “Bệnh đái tháo nhạt do thận”, dạng bệnh tiểu đường mắc phải phổ biến nhất. Các từ khóa viết thường được trả về vì đó là cách chúng được lưu trữ trong chỉ mục khi ta đặt nó qua bộ phân tích tiêu chuẩn khi lập chỉ mục. Chúng ta hoàn toàn không chỉ định bất cứ điều gì trong khi lập chỉ mục, vì vậy bộ phân tích tiêu chuẩn được sử dụng theo mặc định. Các từ khóa thú vị khác trong top 30 là *avp*, một gen liên quan đến bệnh tiểu đường; *thirst* - khát nước, một triệu chứng của bệnh tiểu đường; và *Amilorua*, một loại thuốc cho bệnh tiểu đường. Những từ khóa này dường như mô tả bệnh tiểu đường, nhưng chúng ta đang thiếu các từ khóa nhiều thuật ngữ; chúng ta chỉ lưu trữ các thuật ngữ riêng lẻ trong chỉ mục vì đây là hành vi mặc định. Một số từ sẽ không bao giờ tự xuất hiện vì chúng không được sử dụng thường xuyên nhưng vẫn có ý nghĩa khi được sử dụng kết hợp với các thuật ngữ khác. Hiện tại ta bỏ lỡ mối quan hệ giữa các điều khoản nhất định. Lấy *avp*, ví dụ; nếu như *avp* luôn được viết ở dạng đầy đủ “Nephrogenic Diabetes Insipidus”, nó sẽ không được chọn. Lưu trữ *n-gam* (kết hợp của *n* số lượng từ) chiếm không gian lưu trữ và việc sử dụng chúng cho các truy vấn hoặc tổng hợp sẽ đánh thuế máy chủ tìm kiếm. Quyết định nơi dừng là một bài tập cân bằng và tùy thuộc vào dữ liệu và trường hợp sử dụng của bạn.

Nói chung, bigrams (sự kết hợp của hai thuật ngữ) rất hữu ích vì bigrams có ý nghĩa tồn tại trong ngôn ngữ tự nhiên, mặc dù 10 gam không quá nhiều. Các khái niệm khóa bigram sẽ hữu ích cho việc lập hồ sơ bệnh, nhưng để tạo các tập hợp thuật ngữ có ý nghĩa bigram đó, bạn cần lưu trữ chúng dưới dạng bigrams trong chỉ mục của mình. Như thường thấy trong khoa học dữ liệu, bạn sẽ cần quay lại một số bước để thực hiện một số thay đổi. Hãy quay trở lại giai đoạn chuẩn bị dữ liệu.S

### 6.2.4 Xem lại bước 3: Chuẩn bị dữ liệu cho hồ sơ bệnh tật

Không có gì ngạc nhiên khi bạn quay lại chuẩn bị dữ liệu, như thể hiện trong hình 6.33. Rốt cuộc, quy trình khoa học dữ liệu là một quy trình lặp đi lặp lại. Khi bạn lập chỉ mục dữ liệu của mình, bạn hầu như không làm sạch dữ liệu hoặc chuyển đổi dữ liệu. Ví dụ, bạn có thể thêm tính năng làm sạch dữ liệu bằng cách dừng lọc từ. *Ngưng từ* là những từ phổ biến đến mức chúng thường bị loại bỏ vì chúng có thể gây ô nhiễm kết quả. Chúng tôi sẽ không dừng lọc từ (hoặc làm sạch dữ liệu khác) ở đây, nhưng bạn có thể tự mình thử.

Để lập chỉ mục bigram, bạn cần tạo bộ lọc mã thông báo và bộ phân tích văn bản của riêng mình. Một *token filter* - bộ lọc mã thông báo có khả năng biến đổi mã thông báo. Bộ lọc mã thông báo cụ thể của bạn cần kết hợp các mã thông báo để tạo *n*-grams, còn được gọi là *shingles* –



Hình 6.33 Quy trình khoa học dữ liệu bước 3: chuẩn bị dữ liệu. Làm sạch dữ liệu cho văn bản có thể ngừng lọc từ; chuyển đổi dữ liệu có thể là chữ thường của các ký tự.

*bệnh zona*. Trình mã thông báo Elasticsearch mặc định được gọi là trình mã thông báo tiêu chuẩn và nó sẽ tìm kiếm các ranh giới từ, chẳng hạn như khoảng cách giữa các từ, để cắt văn bản thành các mã thông báo hoặc thuật ngữ khác nhau. Hãy xem các cài đặt mới cho chỉ số bệnh, như được hiển thị trong danh sách sau.

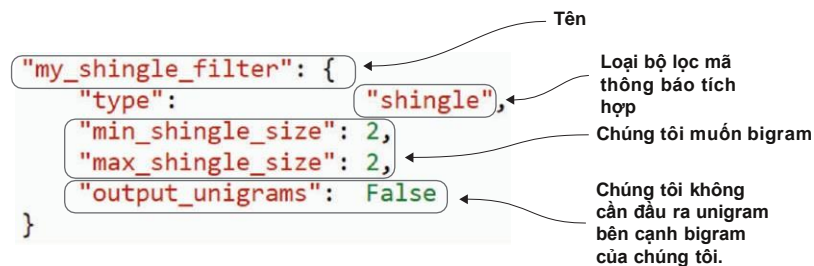
#### Listing 6.5 Cập nhật cài đặt chỉ mục Elasticsearch

```
settings={
  "analysis": {
    "filter": {
      "my_shingle_filter": {
        "type": "shingle",
        "min_shingle_size": 2,
        "max_shingle_size": 2,
        "output_unigrams": False
      }
    },
    "analyzer": {
      "my_shingle_analyzer": {
        "type": "custom",
        "tokenizer": "standard",
        "filter": [
          "lowercase",
          "my_shingle_filter"
        ]
      }
    }
  }
}

client.indices.close(index=indexName)
client.indices.put_settings(index=indexName, body = settings)
client.indices.open(index=indexName)
```

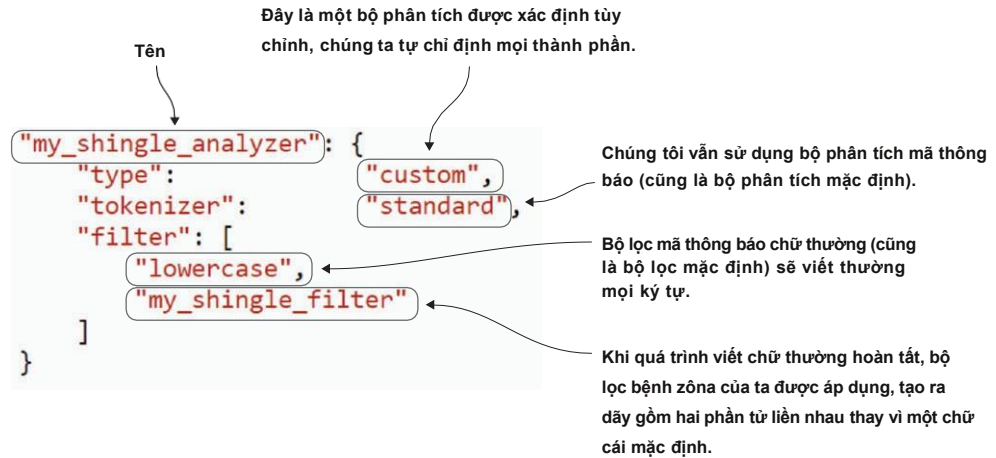
Trước khi bạn có thể thay đổi một số cài đặt nhất định, chỉ mục cần phải được đóng lại. Sau khi thay đổi cài đặt, bạn có thể mở lại chỉ mục.

Bạn tạo hai thành phần mới: bộ lọc mã thông báo có tên là “my shingle filter” và một bộ phân tích mới có tên là “my\_shingle\_analyzer”. Bởi vì  $n$ -grams rất phổ biến, Elasticsearch đi kèm với loại bộ lọc mã thông báo shingle tích hợp sẵn. Tất cả những gì bạn cần nói với nó là bạn muốn bigrams “min\_shingle\_size” : 2, “max\_shingle\_size” : 2, như hình 6.34. Bạn có thể sử dụng các hình bất quá và cao hơn, nhưng với mục đích trình diễn, điều này là đủ.



Hình 6.34 Bộ lọc mã thông báo shingle để tạo bigram

Bộ phân thể hiện trong hình 6.35 là sự kết hợp của tất cả các thao tác cần thiết để chuyển từ văn bản đầu vào sang chỉ mục. Nó kết hợp bộ lọc shingle – bệnh zona, nhưng nó còn hơn thế nữa. Trình mã thông báo chia văn bản thành các mã thông báo hoặc thuật ngữ; sau đó bạn có thể sử dụng bộ lọc chữ thường để không có sự khác biệt khi tìm kiếm “Diabetes” - “Bệnh tiểu đường” so với “diabetes” - “bệnh tiểu đường”. Cuối cùng, bạn áp dụng bộ lọc bệnh zona, tạo nên bigrams.



**Hình 6.35** Một bộ phân tích tùy chỉnh với mã thông báo tiêu chuẩn và bộ lọc mã thông báo zona để tạo ra các dãy gồm hai phần tử liền nhau

Lưu ý rằng bạn cần đóng chỉ mục trước khi cập nhật cài đặt. Sau đó, bạn có thể mở lại chỉ mục một cách an toàn khi biết rằng cài đặt của mình đã được cập nhật. Không phải tất cả các thay đổi cài đặt đều yêu cầu đóng chỉ mục, nhưng thay đổi này thì có. Bạn có thể tìm thấy tổng quan về cài đặt nào cần đóng chỉ mục tại <http://www.elastic.co/guide/en/elasticsearch/reference/current/indices-update-settings.html>.

Chỉ mục hiện đã sẵn sàng để sử dụng máy phân tích mới của bạn. Đối với điều này, bạn sẽ tạo một loại tài liệu mới, disease2, với một ánh xạ mới, như thể hiện trong danh sách sau đây.

#### Listing 6.6 Tạo ánh xạ loại tài liệu Elasticsearch nâng cao hơn

```

docType = 'diseases2'
diseaseMapping = {
  'properties': {
    'name': {'type': 'string'},
    'title': {'type': 'string'},
    'fulltext': {
      "type": "string",
      "fields": {
        "shingles": {
          "type": "string",
          "analyzer": "my_shingle_analyzer"
        }
      }
    }
  }
}

```

Bản đồ bệnh mới khác với bản đồ cũ bằng cách bổ sung trường `fulltext.shingles` chứa dãy gồm hai phần tử liền nhau của bạn.

```

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

```

Ở trong `fulltext` bây giờ bạn có thêm một tham số, `fields`. Tại đây bạn có thể chỉ định tất cả các đồng vị khác nhau của `fulltext`. Bạn chỉ có một; nó đi theo tên shingles và sẽ phân tích `fulltext` với `my_shingle_analyzer` mới của bạn. Bạn vẫn có quyền truy cập vào bản gốc `fulltext` của mình, và bạn không chỉ định máy phân tích cho việc này, vì vậy máy tiêu chuẩn sẽ được sử dụng như trước đây. Bạn có thể truy cập cái mới bằng cách đặt tên thuộc tính theo sau là tên trường của nó: `fulltext.shingles`. Tất cả những gì bạn cần làm bây giờ là thực hiện các bước trước đó và lập chỉ mục dữ liệu bằng API Wikipedia, như được hiển thị trong danh sách sau.

#### Listing 6.7 Lập chỉ mục lại các giải thích về bệnh trên Wikipedia với ánh xạ loại tài liệu mới

```

dl = wikipedia.page("Lists_of_diseases")
diseaseListArray = []
for link in dl.links[15:42]:
    try:
        diseaseListArray.append(wikipedia.page(link))
    except Exception,e:
        print str(e)

```

```

checkList = [["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"]]

```

Danh sách kiểm tra là một mảng chứa "ký tự đầu tiên" được phép. Nếu một căn bệnh không tuân thủ, bạn bỏ qua nó.

Vòng lặp  
qua danh  
sách bệnh

```

for diseaselistNumber, diseaselist in enumerate(diseaseListArray):
    for disease in diseaselist.links: #loop through lists of links for every
        disease list
            try:
                if disease[0] in checkList[diseaselistNumber]
and disease[0:3] != "List":
                    currentPage = wikipedia.page(disease)
                    client.index(index=indexName,
doc_type=docType,id = disease, body={"name": disease,
"title":currentPage.title ,
"fulltext":currentPage.content})
            except Exception,e:
                print str(e)

```

Trước tiên kiểm tra xem đó có phải là bệnh không, sau đó lập chỉ mục cho nó.

Không có gì mới ở đây, chỉ là lần này bạn sẽ lập chỉ mục `doc_type disease2` thay vì `diseases`. Khi quá trình này hoàn tất, bạn lại có thể chuyển sang bước 4, khám phá dữ liệu và kiểm tra kết quả.

### 6.2.5 Xem lại Bước 4: Khám phá dữ liệu để lập hồ sơ bệnh tật

Một lần nữa, bạn đã đến giai đoạn khám phá dữ liệu. Bạn có thể điều chỉnh truy vấn tổng hợp và sử dụng trường mới của mình để cung cấp cho bạn các khái niệm chính liên quan đến bệnh tiểu đường:

**Listing 6.8 Tập hợp các thuật ngữ có ý nghĩa về “tiểu đường” với bigram**

```
searchBody={
  "fields":["name"],
  "query":{
    "filtered" : {
      "filter": {
        "term": {'name':'diabetes'}
      }
    }
  },
  "aggregations" : {
    "DiseaseKeywords" : {
      "significant_terms" : { "field" : "fulltext", "size" : 30 }
    },
    "DiseaseBigrams": {
      "significant_terms" : { "field" : "fulltext.shingles",
        "size" : 30 }
    }
  }
}
client.search(index=indexName,doc_type=docType,
body=searchBody, from_= 0, size=3)
```

Tổng hợp mới của bạn, được gọi là DiseaseBigrams, sử dụng trường `fulltext.shingles` để cung cấp một vài hiểu biết mới về bệnh tiểu đường. Các thuật ngữ chính mới này xuất hiện:

- *Excessive discharge – tiểu quá mức*—Một bệnh nhân tiểu đường cần phải đi tiểu thường xuyên.
- *Causes polyuria - gây đa niệu*—Điều này chỉ ra cùng một điều: bệnh tiểu đường khiến bệnh nhân đi tiểu thường xuyên.
- *Deprivation test - kiểm tra tước đoạt*—Đây thực ra là một bất quá, “thử nghiệm thiếu nước”, nhưng nó được công nhận qua bài kiểm tra vì bạn có bigrams. Đó là xét nghiệm để xác định bệnh nhân có bị tiểu đường hay không.
- *Excessive thirst - khát*—Bạn đã tìm thấy “khát nước” với tìm kiếm từ khóa unigram của mình, nhưng về mặt kỹ thuật tại thời điểm đó, nó có thể có nghĩa là “không khát nước”.

Có những bigram, unigram thú vị khác, và có lẽ cả trigram nữa. Nhìn chung, chúng có thể được sử dụng để phân tích một văn bản hoặc một tập hợp các văn bản trước khi đọc chúng. Lưu ý rằng bạn đã đạt được kết quả mong muốn mà không cần đến giai đoạn lập mô hình. Đôi khi, có ít nhất một lượng thông tin có giá trị bằng nhau được tìm thấy trong quá trình khám phá dữ liệu cũng như trong quá trình lập mô hình dữ liệu. Bây giờ bạn đã hoàn toàn đạt được mục tiêu phụ của mình, bạn có thể chuyển sang bước 6 của quy trình khoa học dữ liệu: trình bày và tự động hóa.

## 6.2.6 Bước 6: Trình bày và tự động hóa

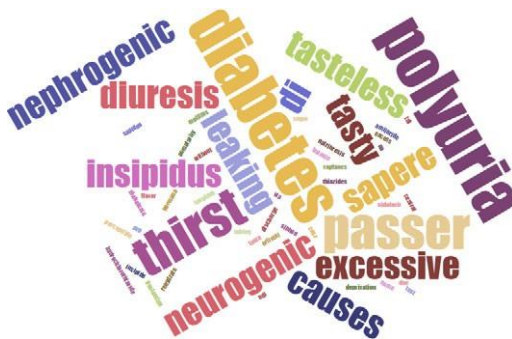
Mục tiêu chính của bạn, chẩn đoán bệnh, đã biến thành một công cụ chẩn đoán tự phục vụ bằng cách cho phép bác sĩ truy vấn nó thông qua một ứng dụng web chẳng hạn. Bạn sẽ không xây dựng một trang web trong trường hợp này, nhưng nếu bạn định làm như vậy, vui lòng đọc thanh bên “Elasticsearch for web applications.” - “Elasticsearch cho các ứng dụng web”.

### Elasticsearch cho các ứng dụng web

Như với bất kỳ cơ sở dữ liệu nào khác, việc hiển thị trực tiếp API REST Elasticsearch của bạn lên giao diện người dùng của các ứng dụng web là một cách làm không tốt. Nếu một trang web có thể trực tiếp thực hiện các yêu cầu POST tới cơ sở dữ liệu của bạn, thì bất kỳ ai cũng có thể dễ dàng xóa dữ liệu của bạn: luôn cần có một lớp trung gian. Lớp giữa này có thể là Python nếu phù hợp với bạn. Hai giải pháp Python phổ biến sẽ là Django hoặc khung Django REST kết hợp với giao diện người dùng độc lập. Django thường được sử dụng để xây dựng *round-trip applications* - ứng dụng khứ hồi (ứng dụng web trong đó máy chủ xây dựng giao diện người dùng một cách linh hoạt, được cung cấp dữ liệu từ cơ sở dữ liệu và hệ thống tạo khuôn mẫu). Khung Django REST là một plugin cho Django, biến Django thành một dịch vụ REST, cho phép nó trở thành một phần của các ứng dụng một trang. Ứng dụng một trang là một ứng dụng web sử dụng một trang web duy nhất làm điểm neo nhưng có khả năng thay đổi nội dung một cách linh hoạt bằng cách truy xuất các tệp tĩnh từ máy chủ HTTP và dữ liệu từ các API RESTful. Cả hai cách tiếp cận (khứ hồi và một trang) đều ổn, miễn là bản thân máy chủ Elasticsearch không mở cửa cho công chúng vì nó không có các biện pháp bảo mật tích hợp. Bảo mật có thể được thêm trực tiếp vào Elasticsearch bằng cách sử dụng “Shield”, một dịch vụ phải trả phí của Elasticsearch.

Mục tiêu phụ, hồ sơ bệnh tật, cũng có thể được đưa đến cấp độ giao diện người dùng; có thể để kết quả tìm kiếm tạo ra một đám mây từ tóm tắt trực quan kết quả tìm kiếm. Chúng ta sẽ không đi xa đến thế trong cuốn sách này, nhưng nếu bạn quan tâm đến việc thiết lập thứ gì đó như thế này trong Python, hãy sử dụng thư viện `word_cloud` (`pip install word_cloud`). Hoặc nếu bạn thích JavaScript hơn, thì `D3.js` là một cách hay. Bạn có thể tìm thấy một triển khai ví dụ tại <http://www.jasondavies.com/wordcloud/#%2F%2Fwww.jasondavies.com%2Fwordcloud%2Fabout%2F>.

Việc thêm các từ khóa của bạn trên trang web dựa trên `D3.js` này sẽ tạo ra một đám mây từ unigram giống như từ được hiển thị trong hình 6.36 có thể được tích hợp vào bản trình



Hình 6.36 Đám mây từ unigram với từ khóa bệnh tiểu đường không trọng số từ Elasticsearch

bày kết quả dự án của bạn. Trong trường hợp này, các thuật ngữ không được tính trọng số theo điểm số của chúng, nhưng nó đã cung cấp một bản trình bày đẹp mắt về các phát hiện.

Có thể có nhiều cải tiến cho ứng dụng của bạn, đặc biệt là trong lĩnh vực chuẩn bị dữ liệu. Nhưng đi sâu vào tất cả các khả năng ở đây sẽ đưa chúng ta đi quá xa; do đó chúng ta đã đi đến cuối chương này. Trong phần tiếp theo, chúng ta sẽ xem xét dữ liệu phát trực tuyến.

## 6.3 Tóm tắt

Trong chương này, bạn đã học được những điều sau:

- *NoSQL* là viết tắt của “Not Only Structured Query Language” - “Không chỉ ngôn ngữ truy vấn có cấu trúc” và phát sinh từ nhu cầu xử lý số lượng và loại dữ liệu ngày càng tăng theo cấp số nhân, cũng như nhu cầu ngày càng tăng đối với các lược đồ đa dạng và linh hoạt hơn như cấu trúc mạng và cấu trúc phân cấp.
- Xử lý tất cả dữ liệu yêu cầu phân vùng cơ sở dữ liệu vì không một máy nào có khả năng thực hiện tất cả công việc. Khi phân vùng, Định lý CAP được áp dụng: bạn có thể có sẵn hoặc nhất quán nhưng không bao giờ có cả hai cùng một lúc.
- Cơ sở dữ liệu quan hệ và cơ sở dữ liệu đồ thị tuân theo các nguyên tắc ACID: tính nguyên tử, tính nhất quán, sự cô lập và độ bền. Cơ sở dữ liệu NoSQL thường tuân theo các nguyên tắc BASE - CƠ SỞ: tính khả dụng cơ bản, trạng thái mềm và tính nhất quán cuối cùng.
- Bốn loại cơ sở dữ liệu NoSQL lớn nhất
  - *Key-value stores* - *Cửa hàng khóa-giá trị*—Về cơ bản là một loạt các cặp khóa-giá trị được lưu trữ trong cơ sở dữ liệu. Các cơ sở dữ liệu này có thể vô cùng lớn và cực kỳ linh hoạt nhưng độ phức tạp của dữ liệu thấp. Một ví dụ nổi tiếng là Redis.
  - *Wide-column databases* - *Cơ sở dữ liệu cột rộng*—Các cơ sở dữ liệu này phức tạp hơn một chút so với các kho lưu trữ keyvalue ở chỗ chúng sử dụng các cột nhưng theo cách hiệu quả hơn so với RDBMS thông thường. Về cơ bản, các cột được tách riêng, cho phép bạn truy xuất dữ liệu trong một cột một cách nhanh chóng. Một cơ sở dữ liệu nổi tiếng là Cassandra.
  - *Document stores* - *Cửa hàng tài liệu*—Những cơ sở dữ liệu này phức tạp hơn một chút và lưu trữ dữ liệu dưới dạng tài liệu. Hiện tại phổ biến nhất là MongoDB, nhưng trong nghiên cứu điển hình của chúng tôi, chúng tôi sử dụng Elasticsearch, đây vừa là kho lưu trữ tài liệu vừa là công cụ tìm kiếm.
  - *Graph databases* - *Cơ sở dữ liệu đồ thị*—Các cơ sở dữ liệu này có thể lưu trữ các cấu trúc dữ liệu phức tạp nhất, vì chúng xử lý các thực thể và mối quan hệ giữa các thực thể một cách thận trọng như nhau. Sự phức tạp này phải trả giá bằng tốc độ tra cứu. Một cái phổ biến là Neo4j, nhưng GraphX (cơ sở dữ liệu đồ thị liên quan đến Apache Spark) đang chiếm ưu thế.
- Elasticsearch là một kho lưu trữ tài liệu và công cụ tìm kiếm toàn văn được xây dựng dựa trên Apache Lucene, công cụ tìm kiếm nguồn mở. Nó có thể được sử dụng để mã hóa, thực hiện truy vấn tổng hợp, thực hiện truy vấn chiều (khía cạnh), truy vấn tìm kiếm hồ sơ, v.v.