7 Sự gia tăng của cơ sở dữ liệu đồ thị

Chương này bao gồm

- Giới thiệu dữ liệu được kết nối và cách nó liên kết đến các biểu đồ và cơ sở dữ liệu biểu đồ
- Tìm hiểu cơ sở dữ liệu đồ thị khác với cơ sở dữ liệu quan hệ thế nào
- Khám phá cơ sở dữ liệu đồ thị Neo4j
- Áp dụng quy trình khoa học dữ liệu cho dự án công cụ đề xuất với cơ sở dữ liệu đồ thị Neo4j

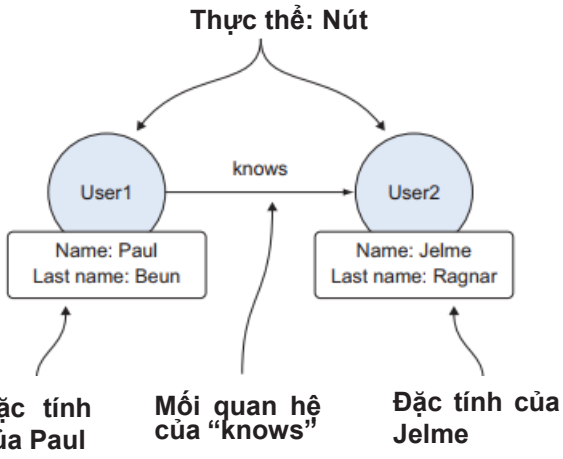
Một mặt, chúng ta đang sản xuất dữ liệu ở quy mô lớn, thúc đẩy những công ty như Google, Amazon và Facebook nghĩ ra những cách thông minh để giải quyết vấn đề này, mặt khác, chúng ta đang phải đối mặt với dữ liệu ngày càng được kết nối với nhau hơn bao giờ hết. Đồ thị và mạng lưới có sức lan tỏa trong cuộc sống của chúng ta. Bằng cách trình bày một số ví dụ thúc đẩy, chúng tôi hy vọng sẽ dạy cho người đọc cách nhận ra một vấn đề đồ thị khi nó tự hiện ra. Trong chương này, chúng ta sẽ xem cách tận dụng các kết nối đó để sử dụng tất cả những gì có giá trị khi sử dụng cơ sở dữ liệu đồ thị, và trình bày cách sử dụng Neo4j, một cơ sở dữ liệu đồ thị phổ biến.

7.1 Giới thiệu cơ sở dữ liệu đồ thị và dữ liệu được kết nối

Hãy bắt đầu bằng cách làm quen với khái niệm dữ liệu được kết nối và biểu diễn của nó dưới dạng dữ liệu biểu đồ.

- Dữ liệu được kết nối – *Connected data*—Như tên cho thấy, dữ liệu được kết nối được đặc trưng bởi thực tế là dữ liệu hiện tại có mối quan hệ làm cho nó được kết nối.
- Đồ thị - *Graphs*—Thường được đề cập đến như dữ liệu được kết nối. Đồ thị rất phù hợp để thể hiện sự kết nối của dữ liệu một cách có ý nghĩa.
- Cơ sở dữ liệu đồ thị - *Graph databases*—Được giới thiệu trong chương 6. Lý do chủ đề này đáng được chú ý đặc biệt là bởi vì, bên cạnh thực tế là dữ liệu đang tăng về kích thước, nó cũng đang trở nên liên kết với nhau nhiều hơn. Không cần nhiều nỗ lực để đưa ra các ví dụ nổi tiếng về dữ liệu được kết nối.

Một ví dụ nổi bật về dữ liệu có dạng mạng là dữ liệu truyền thông xã hội. Phương tiện truyền thông xã hội cho phép chúng ta chia sẻ và trao đổi dữ liệu trong các mạng, từ đó tạo ra một lượng lớn dữ liệu được kết nối. Chúng ta có thể minh họa điều này bằng một ví dụ đơn giản. Giả sử chúng ta có hai người trong dữ liệu của mình, User1 và User2. Hơn nữa, chúng ta biết tên và họ của User1 (Name: Paul và Last name: Beun) và User2 (Name: Jelme và Last name: Ragnar). Một cách tự nhiên để biểu diễn điều này có thể là vẽ nó lên bảng trắng, như thể hiện trong hình 7.1.



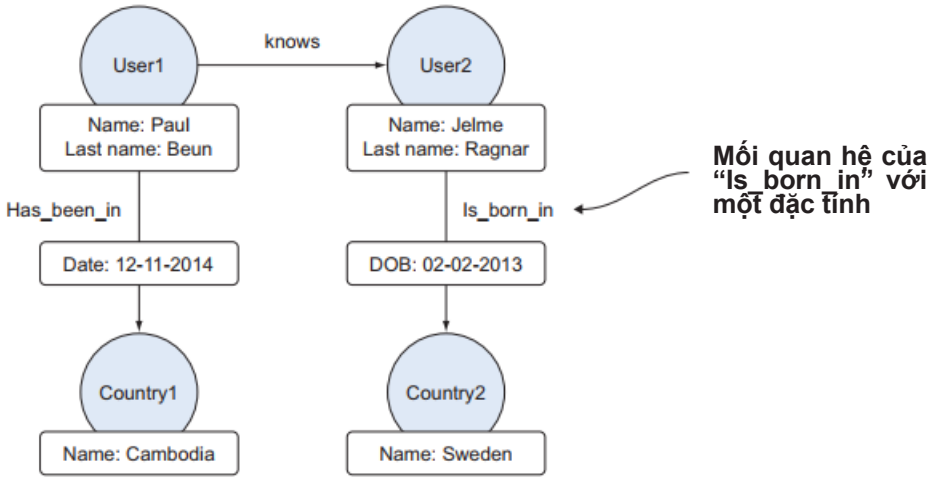
Hình 7.1 Một ví dụ về dữ liệu được kết nối đơn giản: hai thực thể hoặc nút (User1, User2), mỗi thực thể có các thuộc tính (first name, last name), được kết nối bởi một mối quan hệ (knows)

Thuật ngữ của hình 7.1 được mô tả dưới đây:

- Thực thể - *Entities*—Chúng tôi có hai thực thể đại diện cho mọi người (User1 và User2). Các thực thể này có các đặc tính "name" và "lastname".
- Thuộc tính - *Properties*—Các thuộc tính được xác định bởi các cặp khóa-giá trị. Từ biểu đồ này, chúng ta cũng có thể suy ra rằng User1 với đặc tính "name" Paul biết User2 với đặc tính "name" Jelme.

- Mối quan hệ - *Relationships*—Đây là mối quan hệ giữa Paul và Jelme. Lưu ý rằng mối quan hệ có một hướng: Paul là người "knows" Jelme chứ không phải ngược lại. User1 và User2 đều đại diện cho mọi người và do đó có thể được nhóm lại.
- Nhãn - *Labels*—Trong cơ sở dữ liệu đồ thị, người ta có thể nhóm các nút bằng cách sử dụng nhãn. Trong trường hợp này, cả User1 và User2 đều có thể được gắn nhãn là "User".

Dữ liệu được kết nối thường chứa nhiều thực thể và kết nối hơn. Trong hình 7.2, chúng ta có thể thấy một biểu đồ mở rộng hơn. Hai thực thể khác được bao gồm: Country1 với tên Cambodia và Country2 với tên Sweden. Tồn tại thêm hai mối quan hệ: "Has_been_in" và "Is_born_in". Trong biểu đồ trước, chỉ các thực thể bao gồm một thuộc tính, bây giờ các mối quan hệ cũng chứa một thuộc tính. Đồ thị như vậy được gọi là đồ thị thuộc tính. Mối quan hệ kết nối các nút User1 và Country1 thuộc loại "Has_been_in" và có đặc tính "Date" đại diện cho một giá trị dữ liệu. Tương tự, User2 được kết nối với Country2 nhưng thông qua một loại quan hệ khác, thuộc loại "Is_born_in". Lưu ý rằng các loại mối quan hệ cung cấp cho chúng ta ngữ cảnh của mối quan hệ giữa các nút. Các nút có thể có nhiều mối quan hệ.



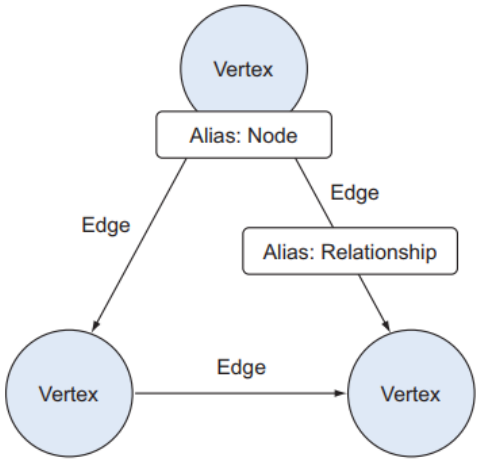
Hình 7.2 Một ví dụ về dữ liệu được kết nối phức tạp hơn trong đó có thêm hai thực thể (Country1 và Country2) và hai mối quan hệ mới ("Has_been_in" và "Is_born_in")

Kiểu biểu diễn dữ liệu này mang lại cho chúng ta một cách trực quan để lưu trữ dữ liệu được kết nối. Để khám phá dữ liệu của chúng ta, cần duyệt biểu đồ theo các đường dẫn được xác định trước để tìm các mẫu mà chúng ta đang tìm kiếm. Nếu một người muốn biết Paul đã đến đâu thì sao? Được dịch sang thuật ngữ cơ sở dữ liệu đồ thị, chúng tôi muốn tìm mẫu "Paul has been in". Để trả lời câu hỏi này, chúng ta sẽ bắt đầu tại nút có

tên “Paul” và đi qua Cambodia thông qua mối quan hệ “Has_been_in”. Do đó, việc duyệt đồ thị, tương ứng với một truy vấn cơ sở dữ liệu, sẽ như sau:

1 Một nút bắt đầu – *A starting node*—Trong trường hợp này, nút có đặc tính tên “Paul”
2 Một đường đi – *A traversal path*—Trong trường hợp này, một con đường bắt đầu từ nút Paul và đi đến Cambodia
3 Nút kết thúc - *End node*—Nút quốc gia có đặc tính tên “Cambodia”

Để hiểu rõ hơn cách cơ sở dữ liệu đồ thị xử lý dữ liệu được kết nối, bạn nên mở rộng thêm một chút về đồ thị nói chung. Đồ thị được nghiên cứu rộng rãi trong các phạm vi của khoa học máy tính và toán học, trong một lĩnh vực được gọi là lý thuyết đồ thị. Lý thuyết đồ thị là sự nghiên cứu về đồ thị, trong đó đồ thị biểu diễn các cấu trúc toán học được sử dụng để mô hình hóa các mối quan hệ theo cặp giữa các đối tượng, như thể hiện trong hình 7.3. Điều khiển chúng trở nên hấp dẫn là chúng có cấu trúc phù hợp để trực quan hóa dữ liệu được kết nối. Một biểu đồ được xác định bởi các đỉnh (còn được gọi là các nút trong thế giới cơ sở dữ liệu đồ thị) và các cạnh (còn được gọi là các mối quan hệ). Những khái niệm này tạo thành các nguyên tắc cơ bản cơ bản mà cấu trúc dữ liệu đồ thị dựa trên.



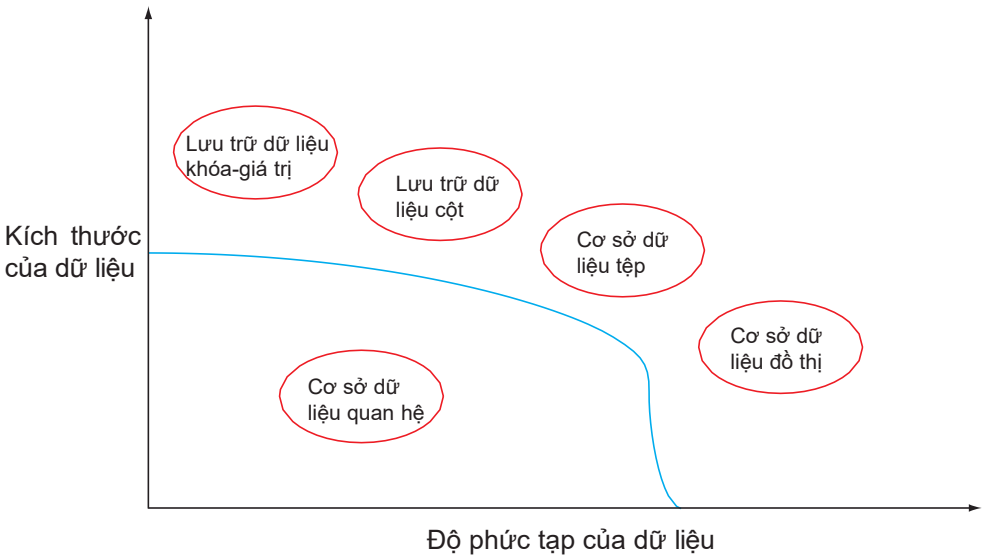
Hình 7.3 Về cốt lõi, một đồ thị bao gồm các nút (còn được gọi là đỉnh) và các cạnh (nối các đỉnh), như đã biết từ định nghĩa toán học của đồ thị. Các bộ sưu tập đối tượng này đại diện cho biểu đồ.

So với các cấu trúc dữ liệu khác, một tính năng đặc biệt của dữ liệu được kết nối là bản chất phi tuyến tính của nó: bất kỳ thực thể nào cũng có thể được kết nối với bất kỳ thực thể nào khác thông qua nhiều loại mối quan hệ cũng như các thực thể và đường dẫn trung gian. Trong đồ thị, bạn có thể thực hiện phân chia giữa đồ thị có hướng và đồ thị vô hướng. Các cạnh của một đồ thị có hướng - làm sao nó có thể khác được - một mũi tên có hướng. Mặc dù người ta có thể lập luận rằng mọi vấn đề bằng cách nào đó có thể được biểu diễn dưới dạng một vấn đề về đồ thị, điều quan trọng là phải hiểu khi nào thì lý tưởng để làm như vậy và khi nào thì không.

7.1.1 Tại sao và khi nào tôi nên sử dụng cơ sở dữ liệu đồ thị?

Nhiệm vụ xác định cơ sở dữ liệu đồ thị nào nên sử dụng có thể là một quá trình phức tạp cần thực hiện. Một khía cạnh quan trọng trong quá trình ra quyết định này là

tìm kiếm đại diện phù hợp cho dữ liệu của bạn. Kể từ đầu những năm 1970, loại cơ sở dữ liệu phổ biến nhất mà người ta phải dựa vào là cơ sở dữ liệu quan hệ. Sau đó, những thứ khác xuất hiện, chẳng hạn như cơ sở dữ liệu phân cấp (ví dụ: IMS) và cơ sở dữ liệu gần nhất của cơ sở dữ liệu đồ thị: cơ sở dữ liệu mạng (ví dụ: IDMS). Nhưng trong những thập kỷ qua, bối cảnh đã trở nên đa dạng hơn nhiều, mang đến cho người dùng cuối nhiều sự lựa chọn hơn tùy thuộc vào nhu cầu cụ thể của họ. Xem xét sự phát triển gần đây của dữ liệu có sẵn, có hai đặc điểm rất phù hợp để làm nổi bật ở đây. Đầu tiên là kích thước của dữ liệu và thứ hai là độ phức tạp của dữ liệu, như thể hiện trong hình 7.4.



Hình 7.4 Hình này minh họa việc định vị cơ sở dữ liệu đồ thị trên không gian hai chiều, trong đó một chiều biểu thị kích thước của dữ liệu đang xử lý và chiều kia biểu thị mức độ phức tạp về cách thức kết nối dữ liệu. Khi cơ sở dữ liệu quan hệ không còn có thể đối phó với sự phức tạp của tập dữ liệu do tính kết nối được của nó chứ không phải kích thước của nó, thì cơ sở dữ liệu đồ thị có thể là lựa chọn tốt nhất của bạn.

Như hình 7.4 chỉ ra, chúng ta sẽ cần dựa vào cơ sở dữ liệu đồ thị khi dữ liệu phức tạp nhưng kích thước vẫn còn nhỏ. Mặc dù “nhỏ” ở đây là một thứ tương đối, nhưng chúng ta vẫn đang nói về hàng trăm triệu nút. Xử lý độ phức tạp là nội dung chính của cơ sở dữ liệu đồ thị và là mục đích cuối cùng "tại sao" bạn sử dụng nó. Để giải thích loại phức tạp nào là có nghĩa là ở đây, trước tiên hãy nghĩ về cách thức hoạt động của một cơ sở dữ liệu quan hệ truyền thống.

Trái ngược với những gì tên của cơ sở dữ liệu quan hệ chỉ ra, không có nhiều quan hệ về chúng ngoại trừ các khóa ngoại và khóa chính là những gì liên quan đến bảng. Ngược lại, các mối quan hệ trong cơ sở dữ liệu đồ thị là công dân hạng nhất. Thông qua khía cạnh này, chúng rất phù hợp để lập mô hình và truy vấn dữ liệu được kết nối. Một

cơ sở dữ liệu quan hệ sẽ cố gắng giảm thiểu sự dư thừa dữ liệu. Quá trình này được gọi là chuẩn hóa cơ sở dữ liệu, trong đó một bảng được phân tách thành các bảng nhỏ hơn (ít dư thừa hơn) trong khi vẫn duy trì nguyên vẹn tất cả thông tin. Trong cơ sở dữ liệu chuẩn hóa, người ta chỉ cần tiến hành thay đổi một thuộc tính trong một bảng. Mục đích của quá trình này là để cô lập các thay đổi dữ liệu trong một bảng. Các hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS) là một lựa chọn tốt làm cơ sở dữ liệu cho dữ liệu phù hợp với định dạng bảng. Các mối quan hệ trong dữ liệu có thể được thể hiện bằng cách nối các bảng. Sự phù hợp của chúng bắt đầu hạ cấp khi phép nối trở nên phức tạp hơn, đặc biệt khi chúng trở thành phép nối nhiều-nhiều. Thời gian truy vấn cũng sẽ tăng lên khi kích thước dữ liệu của bạn bắt đầu tăng và việc duy trì cơ sở dữ liệu sẽ khó khăn hơn. Những yếu tố này sẽ cản trở hiệu suất cơ sở dữ liệu của bạn. Mặt khác, cơ sở dữ liệu đồ thị vốn lưu trữ dữ liệu dưới dạng các nút và mối quan hệ. Mặc dù cơ sở dữ liệu đồ thị được phân loại là một loại cơ sở dữ liệu NoSQL, nhưng vẫn tồn tại xu hướng trình bày chúng dưới dạng một danh mục theo đúng nghĩa của chúng. Người ta tìm kiếm sự biện minh cho điều này bằng cách lưu ý rằng các loại cơ sở dữ liệu NoSQL khác được định hướng tổng hợp, trong khi cơ sở dữ liệu đồ thị thì không.

Ví dụ, một cơ sở dữ liệu quan hệ có thể có một bảng đại diện cho “people” và các đặc tính của họ. Bất kỳ người nào cũng có quan hệ với người khác thông qua quan hệ họ hàng (và tình bạn, v.v.); mỗi hàng có thể đại diện cho một người, nhưng việc kết nối chúng với các hàng khác trong bảng người sẽ là một công việc vô cùng khó khăn. Bạn có thêm một biến chứa mã định danh duy nhất của con đầu tiên và một biến bổ sung để giữ ID của con thứ hai không? Bạn dừng lại ở đâu? Đứa con thứ mười?

Một cách khác là sử dụng bảng trung gian cho các mối quan hệ cha-con, nhưng bạn sẽ cần một bảng riêng cho các loại mối quan hệ khác như tình bạn. Trong trường hợp cuối cùng này, bạn không nhận được sự gia tăng cột mà là sự gia tăng bảng: một bảng quan hệ cho mỗi loại quan hệ. Ngay cả khi bạn thành công bằng cách nào đó trong việc lập mô hình dữ liệu theo cách mà tất cả các mối quan hệ gia đình đều có mặt, bạn sẽ cần những câu hỏi khó để có câu trả lời cho những câu hỏi đơn giản, chẳng hạn như “Tôi muốn các cháu trai của John McBain”. Trước tiên, bạn cần tìm những đứa con của John McBain. Một khi bạn tìm thấy những đứa con của anh ấy, bạn cần tìm những đứa con của họ. Vào thời điểm bạn đã tìm thấy tất cả các cháu trai, bạn đã chạm đến bảng “people” ba lần:

1. Tìm McBain và truy xuất các con của anh ấy.
2. Tra cứu những đứa trẻ ấy bằng ID bạn có và lấy ID con của chúng.
3. Tìm các cháu trai của McBain.

Hình 7.5 cho thấy các tra cứu đệ quy trong một cơ sở dữ liệu quan hệ cần thiết để lấy từ John McBain đến các cháu trai của ông nếu mọi thứ nằm trong một bảng duy nhất.

Hình 7.6 là một cách khác để mô hình hóa dữ liệu: mối quan hệ cha-con là một bảng riêng biệt.

Phải nói rằng các tra cứu đệ quy như thế này là không hiệu quả.

bảng người

First name	Last name	ID	Child ID 1	Child ID 2	Other IDs
John	McBain	1	2	3	…
Wolf	McBain	2	4	5	Null
Arnold	McBain	3	6	7	Null
Moe	McBain	4	Null	Null	Null
Dave	McBain	5	Null	Null	Null
Jago	McBain	6	Null	Null	Null
Carl	McBain	7	Null	Null	Null

❶ Tìm John McBain ❷ Sử dụng ID con để tìm Wolf và Arnold McBain ❸ Sử dụng ID con để tìm Moe, Dave, Jago và Carl McBain

Hình 7.5 Tra cứu đệ quy phiên bản 1: tất cả dữ liệu trong một bảng

People table

First name	Last name	Person ID
John	McBain	1
Wolf	McBain	2
Arnold	McBain	3
Moe	McBain	4
Dave	McBain	5
Jago	McBain	6
Carl	McBain	7

Parent-child relationship table

Parent ID	Child ID
1	2
1	3
2	4
2	5
3	6
3	7

❶ Tìm John McBain ❷ Sử dụng ID con để tìm Wolf và Arnold McBain ❸ Sử dụng ID con để tìm Moe, Dave, Jago và Carl McBain

Hình 7.6 Tra cứu đệ quy phiên bản 2: sử dụng bảng quan hệ cha-con

Cơ sở dữ liệu đồ thị tỏa sáng khi loại phức tạp này xuất hiện. Hãy xem sự phổ biến nhất trong số chúng.

7.2 Giới thiệu Neo4j: cơ sở dữ liệu đồ thị

Dữ liệu được kết nối thường được lưu trữ trong cơ sở dữ liệu đồ thị. Các cơ sở dữ liệu này được thiết kế đặc biệt để đối phó với cấu trúc của dữ liệu được kết nối. Bối cảnh của cơ sở dữ liệu đồ thị có sẵn ngày nay khá đa dạng. Ba cái được biết đến nhiều nhất theo thứ tự

mức độ phổ giảm dần là Neo4j, OrientDb, và Titan. Để giới thiệu nghiên cứu điển hình của mình, chúng tôi sẽ chọn nghiên cứu được biết đến nhiều nhất tại thời điểm viết (xem http://db-engines.com/en/ranking/graph+dbms, tháng 9 năm 2015).

Neo4j là cơ sở dữ liệu đồ thị lưu trữ dữ liệu trong biểu đồ chứa các nút và mối quan hệ (cả hai đều được phép chứa thuộc tính). Loại cơ sở dữ liệu đồ thị này được gọi là đồ thị thuộc tính và rất phù hợp để lưu trữ dữ liệu được kết nối. Nó có một lược đồ linh hoạt cho phép chúng ta tự do thay đổi cấu trúc dữ liệu nếu cần, cung cấp cho chúng ta khả năng thêm dữ liệu mới và các mối quan hệ mới nếu cần. Đó là một dự án nguồn mở, công nghệ hoàn thiện, dễ cài đặt, thân thiện với người dùng và được ghi chép đầy đủ. Neo4j cũng có giao diện dựa trên trình duyệt tạo điều kiện thuận lợi cho việc tạo biểu đồ cho mục đích trực quan hóa. Để tiếp tục, đây sẽ là thời điểm thích hợp để cài đặt Neo4j. Có thể tải xuống Neo4j từ http://neo4j.com/download/. Tất cả các bước cần thiết để cài đặt thành công được tóm tắt trong phụ lục C.

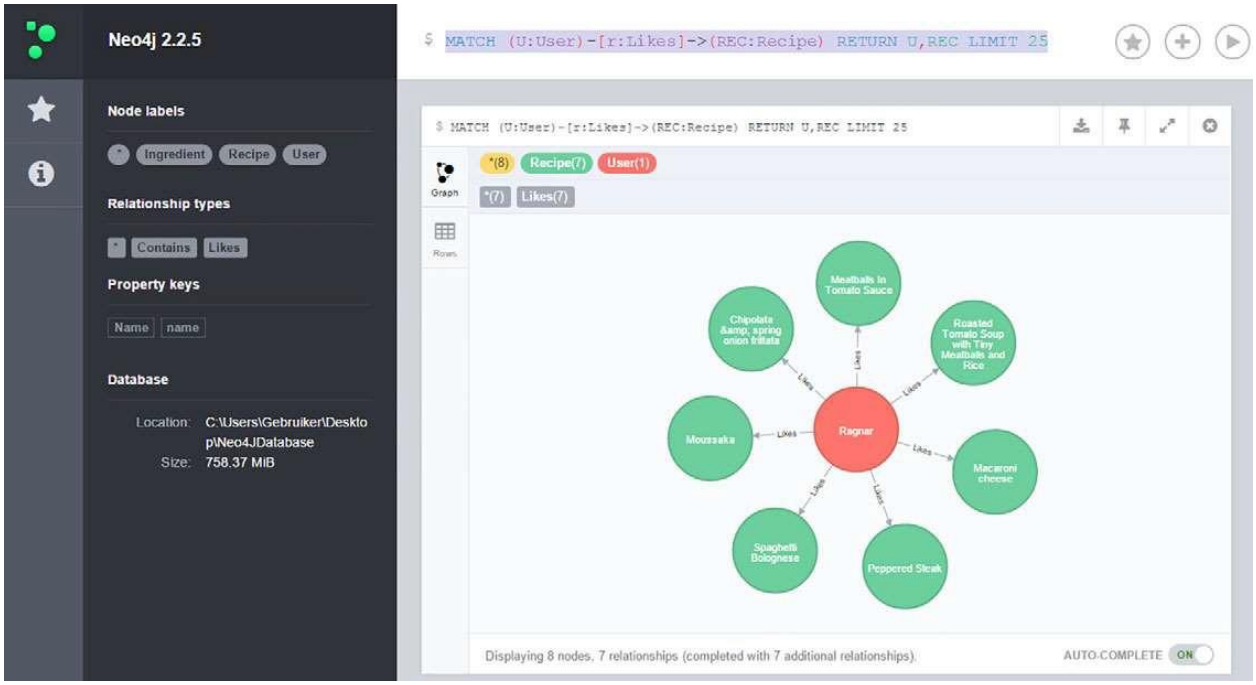
Bây giờ là lúc giới thiệu bốn cấu trúc cơ bản trong Neo4j:

- Nút - *Nodes*—Đại diện cho các thực thể như tài liệu, người dùng, công thức nấu ăn, v.v. Một số thuộc tính có thể được gán cho các nút.
- Mối quan hệ - *Relationships*—Tồn tại giữa các nút khác nhau. Chúng có thể được truy cập độc lập hoặc thông qua các nút mà chúng được gắn vào. Các mối quan hệ cũng có thể chứa các thuộc tính, do đó có tên là mô hình biểu đồ thuộc tính. Mỗi mối quan hệ có một tên và một hướng, cùng nhau cung cấp ngữ cảnh ngữ nghĩa cho các nút được kết nối bởi mối quan hệ.
- Thuộc tính - *Properties*—Cả nút và mối quan hệ đều có thể có thuộc tính. Các thuộc tính được xác định bởi các cặp khóa-giá trị.
- Nhãn - *Labels*—Có thể được sử dụng để nhóm các nút tương tự để tạo điều kiện duyệt các biểu đồ nhanh hơn.

Trước khi tiến hành phân tích, một thói quen tốt là thiết kế cơ sở dữ liệu của bạn một cách cẩn thận để nó phù hợp với các truy vấn mà bạn muốn làm khi thực hiện phân tích của mình. Cơ sở dữ liệu đồ thị có đặc điểm thú vị là chúng thân thiện với bảng trắng. Nếu một người cố gắng vẽ cài đặt vấn đề trên bảng trắng, bản vẽ này sẽ gần giống với thiết kế cơ sở dữ liệu cho vấn đề đã xác định. Do đó, một bản vẽ bảng trắng như vậy sau đó sẽ là điểm khởi đầu tốt để thiết kế cơ sở dữ liệu của chúng ta.

Bây giờ làm thế nào để lấy lại dữ liệu? Để khám phá dữ liệu của mình, chúng ta cần duyệt qua biểu đồ theo các đường dẫn được xác định trước để tìm các mẫu mà chúng ta đang tìm kiếm. Trình duyệt Neo4j là một môi trường lý tưởng để tạo và thử với dữ liệu được kết nối của bạn cho đến khi bạn có được loại biểu diễn phù hợp cho các truy vấn tối ưu, như thể hiện trong hình 7.7. Lược đồ linh hoạt của cơ sở dữ liệu đồ thị rất phù hợp với chúng ta ở đây. Trong trình duyệt này, bạn có thể truy xuất dữ liệu của mình theo hàng hoặc dưới dạng biểu đồ. Neo4j có ngôn ngữ truy vấn riêng để giảm bớt khả năng tạo và truy vấn của biểu đồ.

Cypher là một ngôn ngữ có tính biểu cảm cao, chia sẻ đủ với SQL để nâng cao quá trình học ngôn ngữ này. Trong phần sau, chúng ta sẽ tạo dữ liệu của riêng mình bằng Cypher và chèn dữ liệu đó vào Neo4j. Sau đó, chúng ta có thể thử với dữ liệu.



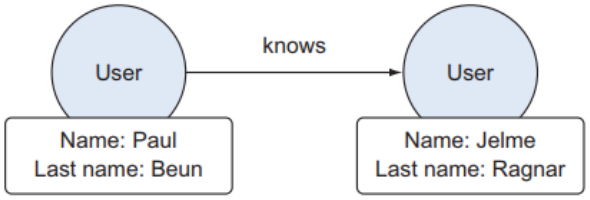
Hình 7.7 Giao diện Neo4j 2.2.5 với truy vấn được giải quyết từ nghiên cứu điển hình của chương

7.2.1 Cypher: ngôn ngữ truy vấn đồ thị

Hãy giới thiệu Cypher và cú pháp cơ bản của nó cho các hoạt động trên đồ thị. Ý tưởng của phần này là trình bày đủ về Cypher để giúp chúng ta bắt đầu sử dụng trình duyệt Neo4j. Ở cuối phần này, bạn sẽ có thể tạo dữ liệu được kết nối của riêng mình bằng Cypher trong trình duyệt Neo4j và chạy các truy vấn cơ bản để truy xuất kết quả của truy vấn. Để có phần giới thiệu chi tiết hơn về Cypher, bạn có thể truy cập http://neo4j.com/docs/stable/cypher-query-lang.html. Chúng ta sẽ bắt đầu bằng cách vẽ một biểu đồ xã hội đơn giản kèm theo một truy vấn cơ bản để lấy một mẫu được xác định trước làm ví dụ. Trong bước tiếp theo, chúng ta sẽ vẽ một biểu đồ phức tạp hơn cho phép chúng ta sử dụng các truy vấn phức tạp hơn trong Cypher. Hơn thế nữa, điều này sẽ giúp chúng ta làm quen với Cypher và đưa trường hợp sử dụng của chúng ta thành hiện thực.

Hình 7.8 cho thấy một biểu đồ xã hội đơn giản gồm hai nút, được kết nối bởi mối quan hệ “knows”. Các nút có cả thuộc tính “name” và “lastname”.

Bây giờ, nếu chúng ta muốn tìm ra khuôn mẫu sau đây, “Paul biết ai?” chúng ta sẽ truy vấn điều này bằng Cypher. Để tìm một mẫu trong Cypher, chúng ta sẽ bắt đầu với một mệnh đề `Match`. Trong



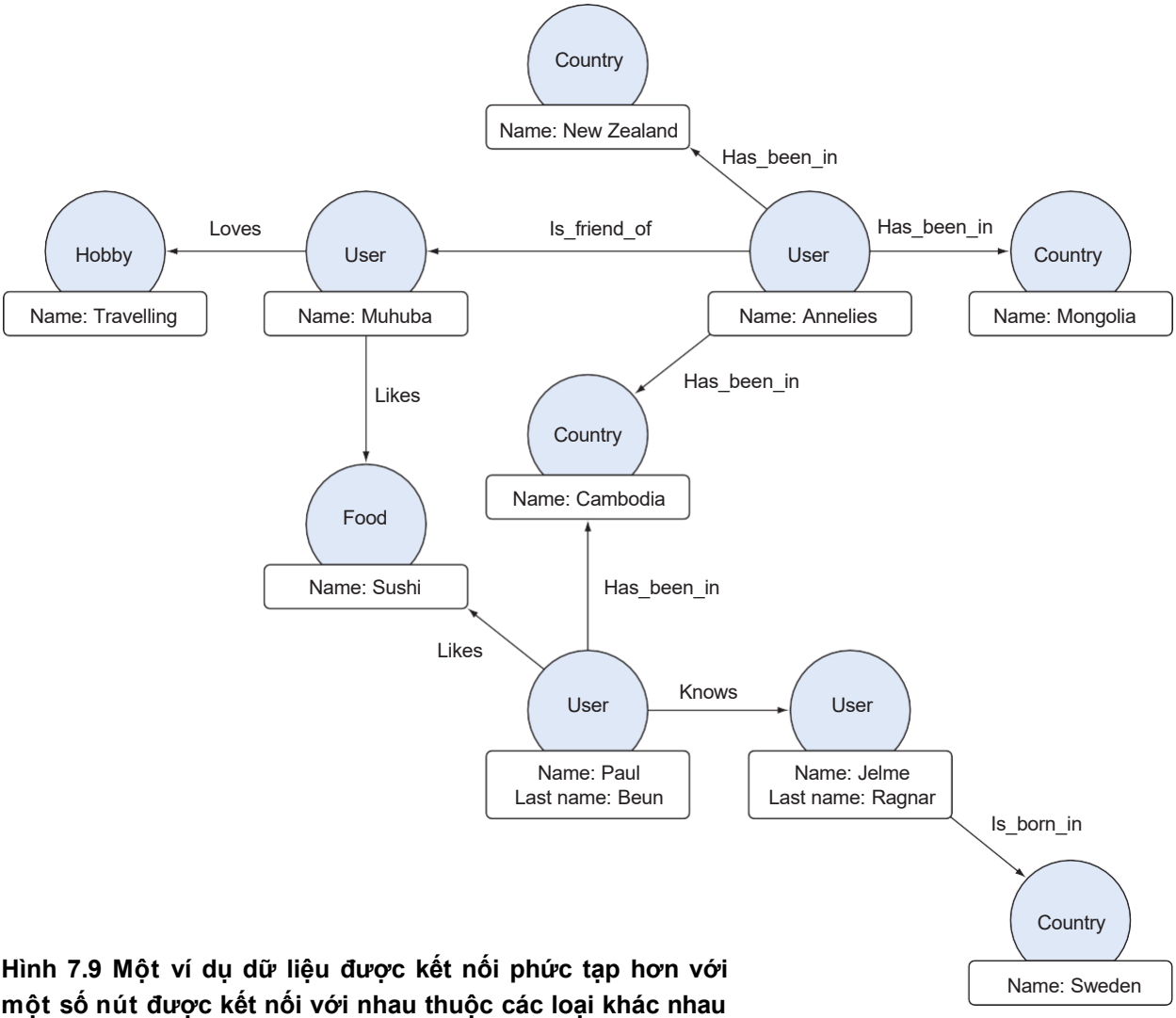
Hình 7.8 Một ví dụ về biểu đồ xã hội đơn giản với hai người dùng và một mối quan hệ

truy vấn này chúng ta sẽ bắt đầu tìm kiếm tại nút User với tên thuộc tính “Paul”. Lưu ý cách nút được đặt trong dấu ngoặc đơn, như được hiển thị trong đoạn mã bên dưới, và mối quan hệ được đặt trong dấu ngoặc vuông. Các mối quan hệ được đặt tên bằng tiền tố dấu hai chấm (:) và hướng được mô tả bằng các mũi tên. Trình giữ chỗ p2 sẽ chứa tất cả các nút User có mối quan hệ kiểu “knows” là một mối quan hệ gửi đến. Với mệnh đề return, chúng ta có thể truy xuất kết quả của truy vấn.

```
Match(p1:User { name: 'Paul' } )-[:knows]->(p2:User)
Return p2.name
```

Lưu ý mối quan hệ chặt chẽ về cách chúng tôi đặt câu hỏi bằng lời nói và cách cơ sở dữ liệu đồ thị chuyển câu hỏi này thành một đường truyền tải. Trong Neo4j, khả năng biểu đạt ấn tượng này có thể thực hiện được nhờ ngôn ngữ truy vấn đồ thị của nó, Cypher.

Để làm cho các ví dụ thú vị hơn, hãy giả sử rằng dữ liệu của chúng ta được biểu thị bằng biểu đồ trong hình 7.9.



Hình 7.9 Một ví dụ dữ liệu được kết nối phức tạp hơn với một số nút được kết nối với nhau thuộc các loại khác nhau

Chúng ta có thể chèn dữ liệu được kết nối trong hình 7.9 vào Neo4j bằng cách sử dụng Cypher. Ta có thể viết các lệnh Cypher trực tiếp trong giao diện dựa trên trình duyệt của Neo 4j hoặc cách khác thông qua trình điều khiển Python (xem http://neo4j.com/developer/python/ để biết tổng quan). Đây là một cách hay để có được cảm giác thực hành với cơ sở dữ liệu đồ thị và dữ liệu được kết nối.

Để viết một câu lệnh tạo phù hợp trong Cypher, trước tiên chúng ta phải hiểu rõ dữ liệu nào chúng ta muốn lưu trữ dưới dạng nút và dữ liệu nào là mối quan hệ, thuộc tính của chúng là gì và liệu nhãn có hữu ích hay không. Quyết định đầu tiên là quyết định dữ liệu nào nên được coi là nút và dữ liệu nào là mối quan hệ để cung cấp ngữ cảnh ngữ nghĩa cho các nút này. Trong hình 7.9, chúng ta đã chọn để đại diện cho những người dùng và quốc gia mà họ đã từng ở dưới dạng các nút. Dữ liệu cung cấp thông tin về một nút cụ thể, chẳng hạn như tên được liên kết với một nút, có thể được biểu diễn dưới dạng một thuộc tính. Tất cả dữ liệu cung cấp ngữ cảnh về hai hoặc nhiều nút sẽ được coi là mối quan hệ. Các nút chia sẻ các đặc điểm chung, chẳng hạn như Cambodia và Sweden đều là các quốc gia, cũng sẽ được nhóm thông qua các nhãn. Trong hình 7.9, điều này đã được thực hiện.

Trong danh sách sau đây, chúng tôi trình bày cách các đối tượng khác nhau có thể được mã hóa trong Cypher thông qua một câu lệnh tạo lớn. Xin lưu ý rằng Cypher phân biệt chữ hoa chữ thường.

Liệt kê 7.1 Câu lệnh tạo dữ liệu Cypher

```
CREATE (user1:User {name :'Annelies'}),
 (user2:User {name :'Paul' , LastName: 'Beun'}),
 (user3:User {name :'Muhuba'}),
 (user4:User {name : 'Jelme' , LastName: 'Ragnar'}),
 (country1:Country { name:'Mongolia'}),
 (country2:Country { name:'Cambodia'}),
 (country3:Country { name:'New Zealand'}),
 (country4:Country { name:'Sweden'}),
 (food1:Food { name:'Sushi' }),
 (hobby1:Hobby { name:'Travelling'}),
 (user1)-[:Has_been_in]->(country1),
 (user1)-[: Has_been_in]->(country2),
 (user1)-[: Has_been_in]->(country3),
 (user2)-[: Has_been_in]->(country2),
 (user1)-[: Is_mother_of]->(user4),
 (user2)-[: knows]->(user4),
 (user1)-[: Is_friend_of]->(user3),
 (user2)-[: Likes]->( food1),
 (user3)-[: Likes]->( food1),
 (user4)-[: Is_born_in]->(country4)
```

Chạy câu lệnh tạo này trong một lần có lợi thế là sự thành công của việc thực thi này sẽ đảm bảo cho chúng ta rằng cơ sở dữ liệu đồ thị đã được tạo thành công. Nếu có lỗi, biểu đồ sẽ không được tạo.

Trong một kịch bản thực tế, ta cũng nên xác định các chỉ mục và ràng buộc để đảm bảo tra cứu nhanh và không phải tìm kiếm toàn bộ cơ sở dữ liệu. Chúng ta chưa làm điều này ở đây vì tập dữ liệu mô phỏng của chúng ta còn nhỏ. Tuy nhiên, điều này có thể dễ dàng thực hiện bằng Cypher. Tham khảo Tài liệu về Cypher để tìm hiểu thêm

Về chỉ mục và ràng buộc (http://neo4j.com/docs/stable/cypherdoc-labels-constraints-and-indexes.html). Bây giờ chúng tôi đã tạo dữ liệu của mình, chúng tôi có thể truy vấn dữ liệu đó. Truy vấn sau đây sẽ trả về tất cả các nút và mối quan hệ trong cơ sở dữ liệu:

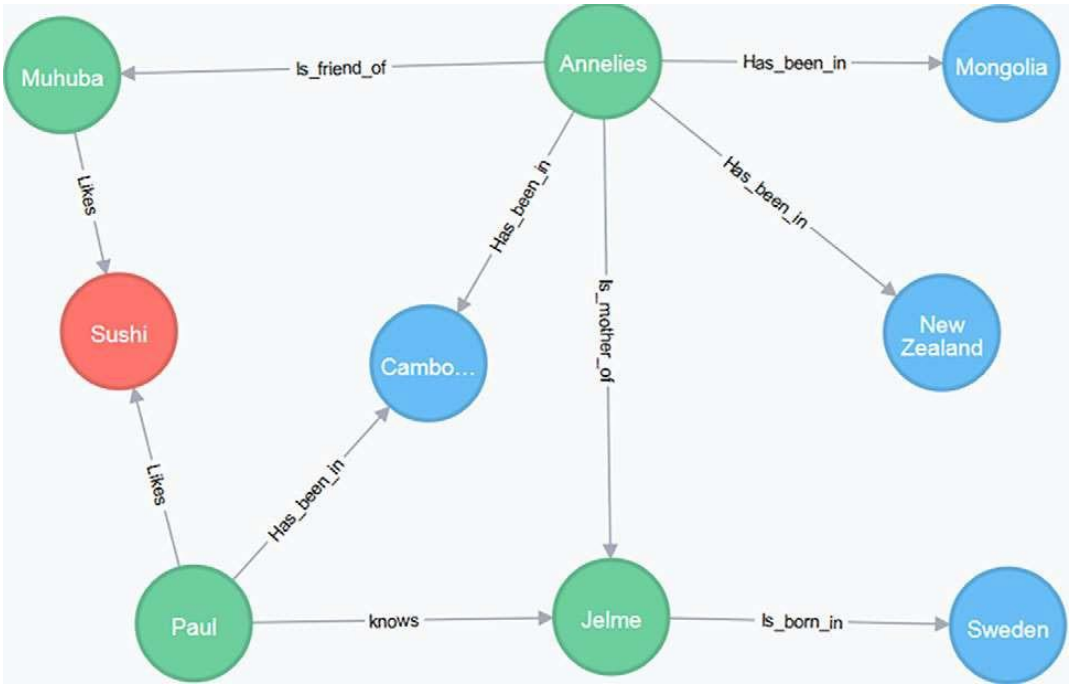
```
MATCH (n)-[r]-()
RETURN n,r
```

Tìm tất cả các nút (n) và tất cả các mối quan hệ của chúng [r].

Hiển thị tất cả các nút n và tất cả các mối quan hệ r.

Hình 7.10 hiển thị cơ sở dữ liệu mà chúng ta đã tạo. Chúng ta có thể so sánh biểu đồ này với biểu đồ mà chúng ta đã hình dung trên bảng trắng của mình. Trên bảng trắng, chúng ta đã nhóm các nút của những người trong nhãn "User" và các nút của các quốc gia trong nhãn "Country". Mặc dù các nút trong hình này không được biểu thị bằng nhãn của chúng, nhưng nhãn vẫn có trong cơ sở dữ liệu của chúng ta. Bên cạnh đó, chúng ta cũng bỏ lỡ một nút ("Hobby") và mối quan hệ kiểu "Loves". Chúng có thể dễ dàng được thêm vào thông qua câu lệnh merge (hợp nhất) để tạo nút và mối quan hệ nếu chúng chưa tồn tại:

```
Merge (user3)-[: Loves]->( hobby1)
```



Hình 7.10 Biểu đồ được vẽ trong hình 7.9 hiện đã được tạo trong giao diện web Neo4j. Các nút không được biểu thị bằng nhãn của chúng mà bằng tên của chúng. Chúng ta có thể suy luận từ biểu đồ rằng chúng ta đang thiếu nhãn Hobby với tên Travelling. Lý do cho điều này là vì chúng ta đã quên bao gồm nút này và mối quan hệ tương ứng của nó trong câu lệnh tạo.

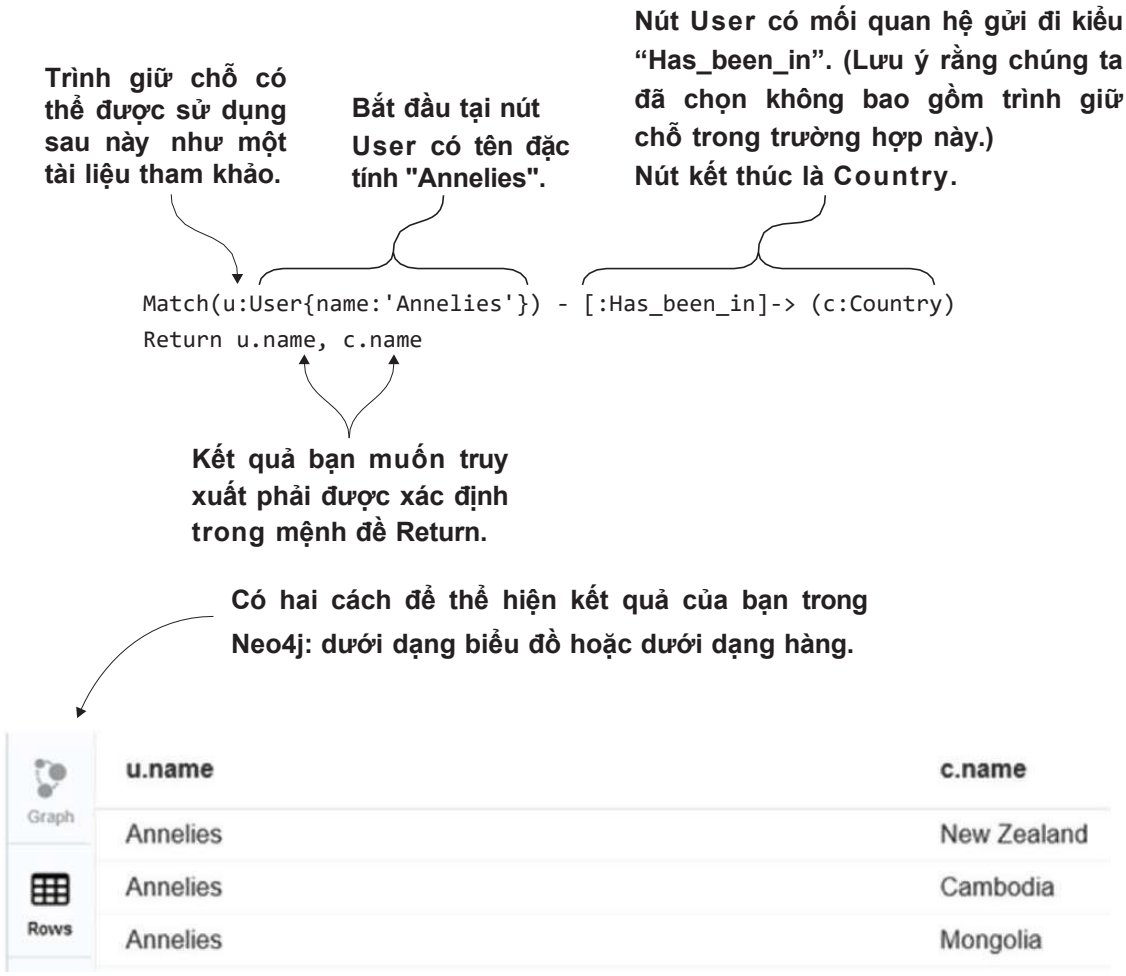
Chúng ta có thể đặt nhiều câu hỏi ở đây. Ví dụ:

- Câu 1: Annelies đã đi thăm những nước nào? Mã Cypher để tạo câu trả lời (hiển thị trong hình 7.11) là

```
Match(u:User{name:'Annelies'}) – [:Has_been_in]-> (c:Country)
Return u.name, c.name
```

- Câu 2: Ai đã ở đâu? Mã Cypher (được giải thích trong hình 7.12) là

```
Match ()-[r: Has_been_in]->()
Return r LIMIT 25
```



Hình 7.11 Kết quả câu 1: Annelies đã đi thăm những nước nào? Chúng ta có thể thấy ba quốc gia mà Annelies đã đến, bằng cách sử dụng bản trình bày theo hàng của Neo4j. Quá trình truyền tải chỉ mất 97 mili giây.

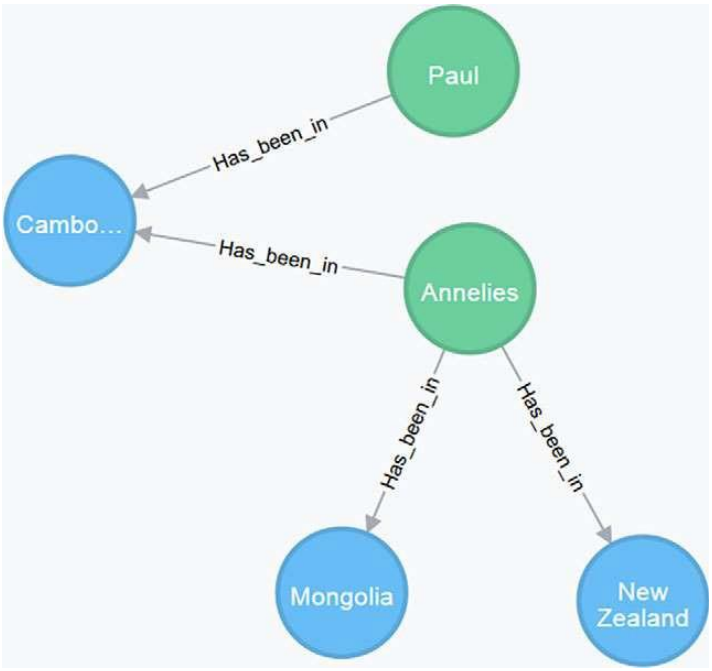
Truy vấn này yêu cầu tất cả các nút có mối quan hệ gửi đi với loại "Has_been_in".

```
MATCH ()-[r:Has_been_in]->()
RETURN r LIMIT 25
```

Các nút kết thúc là tất cả các nút có mối quan hệ đến thuộc loại "Has_been_in".

Hình 7.12 Ai đã ở đâu? Giải thích xây dựng truy vấn.

Khi chạy truy vấn này, chúng ta nhận được câu trả lời như trong hình 7.13.



Hình 7.13 Kết quả câu 2: Ai đã từng ở đâu? Kết quả duyệt của chúng ta hiện được hiển thị trong biểu đồ biểu diễn của Neo4j. Bây giờ chúng ta có thể thấy rằng Paul, ngoài Annelies, cũng đã từng đến Cambodia.

Trong câu hỏi 2, chúng ta đã chọn không chỉ định nút bắt đầu. Do đó, Cypher sẽ đi đến tất cả các nút có trong cơ sở dữ liệu để tìm những nút có mối quan hệ hướng ngoại kiểu "Has_been_in". Bạn nên tránh việc không chỉ định một nút bắt đầu vì tùy thuộc vào kích thước cơ sở dữ liệu của bạn, một truy vấn như vậy có thể mất nhiều thời gian để hội tụ.
Thử với dữ liệu để có được cơ sở dữ liệu đồ thị phù hợp cũng có nghĩa là xóa rất nhiều dữ liệu. Cypher có một câu lệnh xóa phù hợp để xóa một lượng nhỏ dữ liệu.

Truy vấn sau đây trình bày cách xóa tất cả các nút và mối quan hệ trong cơ sở dữ liệu:

```
MATCH(n)
Optional MATCH (n)-[r]-()
Delete n,r
```

Bây giờ chúng ta đã làm quen với dữ liệu được kết nối và có kiến thức cơ bản về cách nó được quản lý trong cơ sở dữ liệu đồ thị, chúng ta có thể tiến thêm một bước và xem xét các ứng dụng thực, trực tiếp của dữ liệu được kết nối. Ví dụ, một biểu đồ xã hội có thể được sử dụng để tìm các cụm nút được kết nối chặt chẽ bên trong các nhóm biểu đồ. Những người trong một cụm không biết nhau sau đó có thể được giới thiệu với nhau. Khái niệm tìm kiếm các nút được kết nối chặt chẽ, các nút có nhiều đặc điểm chung, là một khái niệm được sử dụng rộng rãi.
Trong phần tiếp theo, chúng ta sẽ sử dụng ý tưởng này, với mục đích là tìm các cụm bên trong mạng lưới thành phần.

7.3 Ví dụ về dữ liệu được kết nối: Công cụ đề xuất công thức nấu ăn

Một trong những trường hợp sử dụng phổ biến nhất cho cơ sở dữ liệu đồ thị là sự phát triển của các công cụ đề xuất. Các công cụ đề xuất đã được áp dụng rộng rãi thông qua lời hứa tạo ra nội dung có liên quan. Sống trong thời đại với lượng dữ liệu dồi dào như vậy có thể khiến nhiều người tiêu dùng choáng ngợp. Doanh nghiệp thấy rõ nhu cầu sáng tạo trong cách thu hút khách hàng thông qua nội dung được cá nhân hóa, từ đó tận dụng thế mạnh của công cụ giới thiệu.

Trong nghiên cứu điển hình của mình, chúng tôi sẽ đề xuất các công thức nấu ăn dựa trên sở thích món ăn của người dùng và mạng lưới nguyên liệu. Trong quá trình chuẩn bị dữ liệu, chúng tôi sẽ sử dụng Elaticsearch để đẩy nhanh quá trình và cho phép tập trung hơn vào cơ sở dữ liệu đồ thị thực tế. Mục đích chính của nó ở đây là thay thế danh sách thành phần của dữ liệu “bẩn” tải xuống bằng thành phần từ danh sách “sạch” của chính chúng ta.

Nếu bạn đã bỏ qua chương này, thì ít nhất bạn nên đọc phụ lục A về cách cài đặt Elaticsearch để bạn có thể chạy nó trên máy tính của mình. Bạn luôn có thể tải xuống chỉ mục mà chúng tôi sẽ sử dụng từ trang tải xuống Manning cho chương này và dán nó vào thư mục dữ liệu Elaticsearch cục bộ của bạn nếu bạn không muốn bận tâm đến nghiên cứu điển hình của chương 6.

Bạn có thể tải xuống thông tin sau từ trang web Manning cho chương này:

Ba tệp mã .py và các đối tác .ipynb của chúng

- *Data Preparation Part 1*—Sẽ tải dữ liệu lên Elaticsearch (ngoài ra, bạn có thể dán chỉ mục có thể tải xuống vào thư mục dữ liệu Elaticsearch cục bộ của mình)
- *Data Preparation Part 2*—Sẽ chuyển dữ liệu từ Elaticsearch sang Neo4j
- Hệ thống thăm dò & đề xuất

Ba tệp dữ liệu

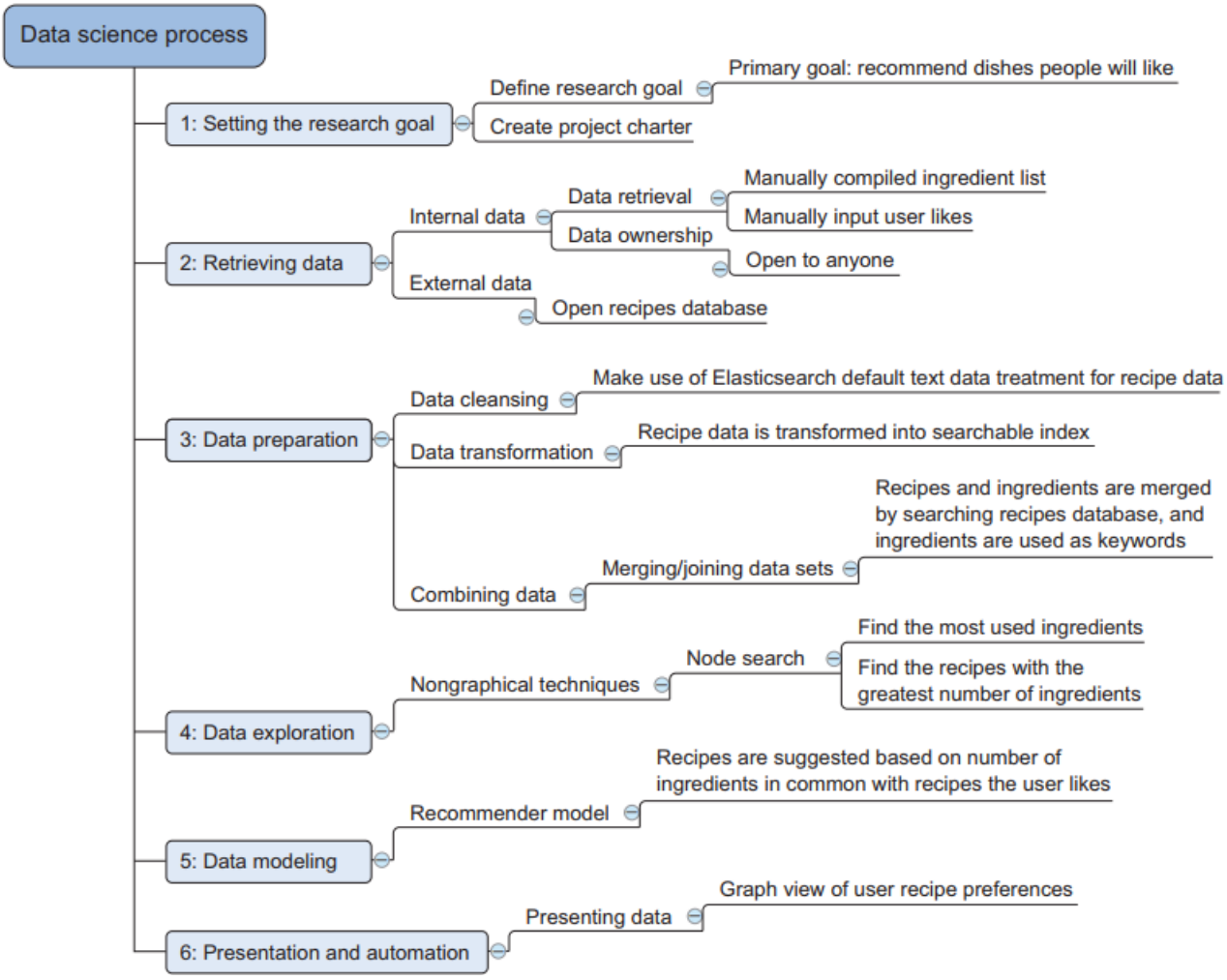
- *Ingredients (.txt)*—Hồ sơ thành phần tự biên soạn
- *Recipes (.json)*—Chứa tất cả các thành phần
- *Elasticsearch index (.zip)*—Chứa chỉ mục Elaticsearch “gastronomical” - “ẩm thực” mà bạn có thể sử dụng để bỏ qua khâu chuẩn bị dữ liệu phần 1

Bây giờ chúng ta đã có mọi thứ mình cần, hãy xem xét mục tiêu nghiên cứu và các bước chúng ta cần thực hiện để đạt được mục tiêu đó.

7.3.1 Bước 1: Đặt mục tiêu nghiên cứu

Hãy xem điều gì sẽ xảy ra khi chúng ta tuân theo quy trình khoa học dữ liệu (hình 7.14).

Mục tiêu chính của chúng tôi là thiết lập một công cụ giới thiệu giúp người dùng của một trang web nấu ăn tìm được công thức phù hợp. Một người dùng có thể thích một số công thức nấu ăn và chúng tôi sẽ căn cứ vào



Hình 7.14 Tổng quan về quy trình khoa học dữ liệu được áp dụng cho mô hình đề xuất dữ liệu được kết nối

các đề xuất món ăn về sự chồng chéo của các thành phần trong mạng công thức nấu ăn. Đây là một cách tiếp cận đơn giản và trực quan, nhưng đã mang lại kết quả khá chính xác. Hãy xem xét ba yếu tố dữ liệu mà chúng tôi yêu cầu.

7.3.2 Bước 2: Truy xuất dữ liệu

Đối với bài tập này, chúng tôi yêu cầu ba loại dữ liệu:

- Công thức nấu ăn và các thành phần tương ứng của họ
- Một danh sách các nguyên liệu riêng biệt mà chúng tôi muốn mô hình hóa
- Ít nhất một người dùng và sở thích của người ấy đối với một số món ăn nhất định

Như mọi khi, chúng ta có thể chia dữ liệu này thành dữ liệu được tạo hoặc có sẵn nội bộ và dữ liệu có được từ bên ngoài.

- Dữ liệu nội bộ - *Internal data*—chúng tôi không có bất kỳ sở thích của người dung hoặc các nguyên liệu xung quanh, nhưng đây là phần nhỏ nhất trong dữ liệu của chúng tôi và dễ dàng tạo ra. Một vài tùy chọn nhập thủ công là đủ để tạo đề xuất. Người dùng nhận được nhiều kết quả thú vị và chính xác hơn khi người đó đưa ra nhiều phản hồi hơn. Chúng tôi sẽ nhập tùy chọn người dùng sau đó trong nghiên cứu điển hình. Một danh sách các thành phần có thể được biên soạn thủ công và sẽ vẫn phù hợp trong nhiều năm tới, vì vậy, hãy thoải mái sử dụng danh sách trong tài liệu có thể tải xuống cho bất kỳ mục đích nào, kể cả thương mại hay mục đích khác.
- Dữ liệu bên ngoài – *External data*—Công thức nấu ăn - *Recipes* là một vấn đề khác nhau. Có hàng nghìn nguyên liệu nhưng chúng có thể được kết hợp thành hàng triệu món ăn. Tuy nhiên, chúng tôi rất may mắn vì một danh sách khá lớn được cung cấp miễn phí tại https://github.com/fictivekin/openrecipes. Rất cảm ơn Fictive Kin vì bộ dữ liệu quý giá này với hơn một trăm nghìn công thức nấu ăn. Chắc chắn ta có các bản sao ở đây, nhưng chúng sẽ không ảnh hưởng xấu đến trường hợp sử dụng của chúng ta.

Hiện tại, chúng ta có hai tệp dữ liệu: danh sách hơn 800 nguyên liệu (ingredients.txt) và hơn một trăm nghìn công thức nấu ăn trong tệp công thức nấu ăn recipes.json. Một mẫu của danh sách nguyên liệu có thể được nhìn thấy trong danh sách sau đây.

Liệt kê 7.2 Mẫu tệp văn bản danh sách thành phần

```
Ditalini
Egg Noodles
Farfalle
Fettuccine
Fusilli
Lasagna
Linguine
Macaroni
Orzo
```

Tệp JSON “openrecipes” chứa hơn một trăm nghìn công thức nấu ăn với nhiều thuộc tính như ngày xuất bản, vị trí nguồn, thời gian chuẩn bị, mô tả, v.v.

Chúng tôi chỉ quan tâm đến tên và danh sách nguyên liệu. Một công thức mẫu được hiển thị trong danh sách sau đây.

Liệt kê 7.3 Một công thức JSON mẫu

```
{ "_id" : { "$oid" : "5160756b96cc62079cc2db15" },
"name" : "Drop Biscuits and Sausage Gravy",
"ingredients" : "Biscuits\n3 cups All-purpose Flour\n2 Tablespoons Baking Powder\n1/2 teaspoon Salt\n1-1/2 stick (3/4 Cup) Cold Butter, Cut Into Pieces\n1-1/4 cup Butermilk\n SAUSAGE GRAVY\n1 pound Breakfast Sausage, Hot Or Mild\n1/3 cup All-purpose Flour\n4 cups Whole Milk\n1/2 teaspoon Seasoned Salt\n2 teaspoons Black Pepper, More To Taste",
"url" : "http://thepioneerwoman.com/cooking/2013/03/drop-biscuits-andsausage-gravy/",
"image" : "http://static.thepioneerwoman.com/cooking/files/2013/03/bisgrav.jpg",
"ts" : { "$date" : 1365276011104 },
"cookTime" : "PT30M",
"source" : "thepioneerwoman",
"recipeYield" : "12",
"datePublished" : "2013-03-11",
"prepTime" : "PT10M",
"description" : "Late Saturday afternoon, after Marlboro Man had returned
home with the soccer-playing girls, and I had returned home with the..."
}
```

Vì chúng ta đang xử lý dữ liệu văn bản ở đây nên vấn đề có hai mặt: thứ nhất, chuẩn bị dữ liệu văn bản như được mô tả trong chương khai phá văn bản. Sau đó, khi dữ liệu được làm sạch hoàn toàn, nó có thể được sử dụng để đưa ra các đề xuất công thức dựa trên mạng lưới các nguyên liệu. Chương này không tập trung vào việc chuẩn bị dữ liệu văn bản vì điều này đã được mô tả ở nơi khác, vì vậy chúng ta sẽ tự cho phép mình sử dụng một phím tắt trong quá trình chuẩn bị dữ liệu sắp tới.

7.3.3 Bước 3: Chuẩn bị dữ liệu

Bây giờ ta có hai tệp dữ liệu theo ý của mình và ta cần kết hợp chúng thành một cơ sở dữ liệu đồ thị. Dữ liệu công thức nấu ăn "bẩn" đặt ra một vấn đề mà chúng ta có thể giải quyết bằng cách sử dụng danh sách nguyên liệu sạch và việc sử dụng công cụ tìm kiếm cũng như cơ sở dữ liệu NoSQL Elaticsearch. Chúng ta đã dựa vào Elaticsearch trong chương trước và bây giờ nó sẽ làm sạch hoàn toàn dữ liệu công thức cho chúng ta khi nó tạo một chỉ mục. Sau đó, chúng tôi có thể tìm kiếm dữ liệu này để liên kết từng nguyên liệu với mọi công thức mà nó xuất hiện. Chúng ta có thể làm sạch dữ liệu văn bản bằng cách sử dụng Python thuần túy, như chúng ta đã làm trong chương khai thác văn bản, nhưng điều này cho thấy thật tốt khi nhận thức được các điểm mạnh của từng cơ sở dữ liệu NoSQL; đừng bó buộc mình vào một công nghệ duy nhất mà hãy sử dụng chúng cùng nhau vì lợi ích của dự án.

Hãy bắt đầu bằng cách nhập dữ liệu công thức của chúng ta vào Elaticsearch. Nếu bạn không hiểu chuyện gì đang xảy ra, vui lòng xem lại nghiên cứu điển hình của chương 6 và nó sẽ trở nên rõ ràng. Đảm bảo bật phiên bản Elaticsearch cục bộ của bạn và kích hoạt môi trường Python với mô-đun Elaticsearch được cài đặt trước khi chạy mã

đoạn trích trong danh sách sau đây. Bạn không nên chạy mã này "nguyên trạng" trong Ipython (hoặc Jupyter) vì nó in mọi khóa công thức ra màn hình và trình duyệt của bạn chỉ có thể xử lý rất nhiều đầu ra. Tắt các câu lệnh in hoặc chạy trong một IDE Python khác. Mã trong đoạn mã này có thể được tìm thấy trong "Data Preparation Part 1.py".

Liệt kê 7.4 Nhập dữ liệu công thức vào Elaticsearch

```
from elasticsearch import Elasticsearch
import json

client = Elasticsearch ()
indexName = "gastronomical"
docType = 'recipes'

client.indices.create(index=indexName)

file_name = 'C:/Users/Gebruiker/Downloads/recipes.json'

recipeMapping = {
        'properties': {
            'name': {'type': 'string'},
            'ingredients': {'type': 'string'}
        }
}

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

with open(file_name, encoding="utf8") as data_file:
    recipeData = json.load(data_file)

for recipe in recipeData:
    print recipe.keys()
    print recipe['_id'].keys()
    client.index(index=indexName,
        doc_type=docType,id = recipe['_id']['$oid'],
        body={"name": recipe['name'], "ingredients":recipe['ingredients']})
```

Nhập mô-đun.

Ứng dụng khách Elaticsearch được sử dụng để giao tiếp với cơ sở dữ liệu.

Tạo chỉ mục.

Vị trí của tập công thức JSON: thay đổi điều này để phù hợp với thiết lập của riêng bạn!

Lập bản đồ cho Elaticsearch với loại tài liệu "recipe".

Tải tệp công thức JSON vào bộ nhớ. Một cách khác để làm điều này là: recipeData = []
with open(file_name) as f:
for line in f:
recipeData.append(json.loads(line))

Chỉ mục công thức nấu ăn. Chỉ tên và nguyên liệu là quan trọng đối với trường hợp sử dụng của chúng tôi. Trong trường hợp một xảy ra sự cố hết thời gian chờ, có thể tăng độ trễ thời gian chờ bằng cách chỉ định, ví dụ, timeout=30 làm đối số.

Nếu mọi thứ suôn sẻ, giờ đây chúng ta có một chỉ mục Elaticsearch với tên "gastronomical" được phổ biến bởi hàng nghìn công thức nấu ăn. Lưu ý rằng chúng tôi đã cho phép các bản sao của cùng một công thức bằng cách không chỉ định tên của công thức làm khóa tài liệu. Nếu cho ví dụ, một công thức được gọi là "lasagna" thì đây có thể là lasagna

cá hồi, lasagna thịt bò, lasagna thịt gà hoặc bất kỳ loại nào khác. Không có công thức nào được chọn làm món lasagna nguyên mẫu; tất cả chúng đều được tải lên Elaticsearch dưới cùng một tên: “lasagna”. Đây là một sự lựa chọn, vì vậy hãy thoải mái khi quyết định. Nó sẽ có tác động đáng kể, như chúng ta sẽ thấy sau này. Cánh cửa hiện đang mở để tải lên một cách có hệ thống cơ sở dữ liệu đồ thị cục bộ của chúng tôi. Đảm bảo phiên bản cơ sở dữ liệu đồ thị cục bộ của bạn được bật khi áp dụng đoạn mã sau. Tên người dùng của chúng tôi cho cơ sở dữ liệu này mặc định là Neo4j và mật khẩu là Neo4ja; đảm bảo điều chỉnh điều này cho thiết lập cục bộ của bạn. Đối với điều này, chúng tôi cũng sẽ yêu cầu thư viện Python dành riêng cho Neo4j có tên là py2neo. Nếu bạn chưa cài đặt, bây giờ sẽ là lúc cài đặt nó vào môi trường ảo của bạn bằng cách sử dụng câu lệnh pip install py2neo hoặc conda install py2neo khi sử dụng Anaconda. Một lần nữa, xin lưu ý rằng mã này sẽ làm hỏng trình duyệt của bạn khi chạy trực tiếp trong Ipython hoặc Jupiter. Bạn có thể tìm thấy mã trong danh sách này trong “Data Preparation Part 2.py”.

Liệt kê 7.5 Sử dụng chỉ mục Elaticsearch để điền vào cơ sở dữ liệu đồ thị

```
from elasticsearch import Elasticsearch
from py2neo import Graph, authenticate, Node, Relationship

client = Elasticsearch ()
indexName = "gastronomical"
docType = 'recipes’

authenticate("localhost:7474", "user", "password")
graph_db = Graph("http://localhost:7474/db/data/")

filename = 'C:/Users/Gebruiker/Downloads/ingredients.txt'
ingredients =[]
with open(filename) as f:
    for line in f:
        ingredients.append(line.strip())

print ingredients
ingredientnumber = 0
grandtotal = 0
for ingredient in ingredients:

    try:
        IngredientNode = graph_db.merge_one("Ingredient","Name",ingredient)
    except:
        continue

ingredientnumber +=1
searchbody = {
        "size" : 99999999,
        "query": {
            "match_phrase":
                {
                    "ingredients":{
```

Nhập mô-đun

Ứng dụng khách Elasticsearch dùng để giao tiếp với cơ sở dữ liệu

Xác thực với tên người dùng của riêng bạn và mật khẩu

Thực thể cơ sở dữ liệu đồ thị

Tệp văn bản nguyên liệu được tải vào bộ nhớ

Thoát vì bạn nhận được /n nếu không đọc từ .txt

Vòng lặp qua các nguyên liệu và lấy kết quả Elaticsearch

Tạo nút trong cơ sở dữ liệu đồ thị cho nguyên liệu hiện tại

Kết hợp cụm từ được sử dụng, vì một số nguyên liệu bao gồm nhiều từ

```
                    "query":ingredient,
                }
            }
        }
    }
result = client.search(index=indexName,doc_type=docType,body=searchbody)

print ingredient
print ingredientnumber
print "total: " + str(result['hits']['total'])

grandtotal = grandtotal + result['hits']['total']
print "grand total: " + str(grandtotal)
for recipe in result['hits']['hits']:
    try:
        RecipeNode =
graph_db.merge_one("Recipe","Name",recipe['_source']['name'])
        NodesRelationship = Relationship(RecipeNode, "Contains",
IngredientNode)
        graph_db.create_unique(NodesRelationship)
        print "added: " + recipe['_source']['name'] + " contains " +
Ingredient
    except:
        continue
print "************************************"
```

Vòng lặp qua công thức nấu ăn được tìm thấy cho nguyên liệu đặc biệt này

Tạo nên mối quan hệ giữa công thức và nguyên liệu

Tạo nút cho mỗi công thức chưa có trong cơ sở dữ liệu đồ thị

Tuyệt vời, giờ đây chúng tôi tự hào là chủ sở hữu của một cơ sở dữ liệu đồ thị chứa đầy các công thức nấu ăn! Đã đến lúc

khám phá dữ liệu được kết nối.

7.3.4 Bước 4: Khám phá dữ liệu

Bây giờ chúng tôi có dữ liệu của mình ở nơi chúng tôi muốn, chúng tôi có thể khám phá thủ công dữ liệu đó bằng giao diện Neo4j tại http://localhost:7474/browser/.

Không có gì ngăn cản bạn chạy mã Cypher của mình trong môi trường này, nhưng Cypher cũng có thể được thực thi thông qua thư viện py2neo. Một câu hỏi thú vị mà chúng ta có thể đặt ra là nguyên liệu nào xuất hiện nhiều nhất trong tất cả các công thức nấu ăn? Những gì có khả năng xâm nhập vào hệ thống tiêu hóa của chúng ta nhất nếu chúng ta chọn ngẫu nhiên và ăn các món ăn từ cơ sở dữ liệu này?

```
from py2neo import Graph, authenticate, Node, Relationship
authenticate("localhost:7474", "user", "password")
graph_db = Graph("http://localhost:7474/db/data/")graph_db.cypher.execute("
    MATCH (REC:Recipe)-[r:Contains]->(ING:Ingredient) WITH ING, count(r) AS num
    RETURN ING.Name as Name, num ORDER BY num DESC LIMIT 10;")
```

Truy vấn được tạo trong Cypher và cho biết: đối với tất cả các công thức nấu ăn và nguyên liệu của chúng, hãy đếm số lượng mối quan hệ trên mỗi nguyên liệu và trả về mười nguyên liệu có nhiều mối quan hệ nhất và số lượng tương ứng của chúng. Kết quả được thể hiện trong hình 7.15.

Hầu hết danh sách top 10 trong hình 7.15 không có gì đáng ngạc nhiên. Với muối đứng đầu danh sách của chúng tôi một cách tự hào, chúng tôi không ngạc nhiên khi thấy các bệnh về mạch máu là kẻ giết người số một ở hầu hết các nước phương Tây. Một câu hỏi thú vị khác xuất hiện trong đầu tôi bây giờ là từ một góc nhìn khác: công thức nấu ăn nào cần nhiều nguyên liệu nhất?

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

Hình 7.15 Top 10 thành phần xuất hiện trong hầu hết các công thức nấu ăn

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("
    MATCH (REC:Recipe)-[r:Contains]->(ING:Ingredient) WITH REC, count(r) AS num
    RETURN REC.Name as Name, num ORDER BY num DESC LIMIT 10;")
```

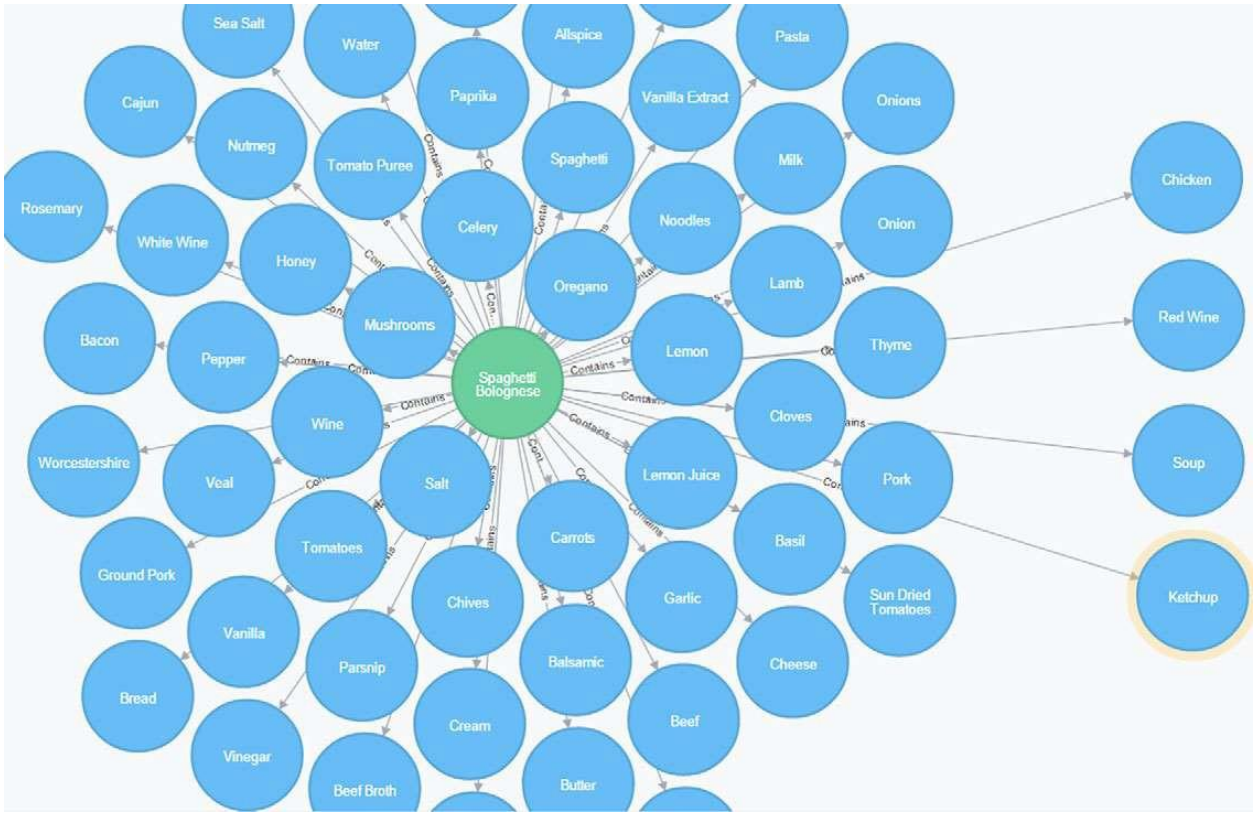
Truy vấn gần giống như trước đây, nhưng thay vì trả về nguyên liệu, chúng tôi yêu cầu công thức nấu ăn. Kết quả được hình 7.16.

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

Hình 7.16 Top 10 món ăn có thể được tạo ra với sự đa dạng nhất về nguyên liệu

Bây giờ đây có thể là một cảnh tượng đáng ngạc nhiên. Spaghetti Bolognese hầu như không phải là loại món ăn cần đến 59 nguyên liệu. Chúng ta hãy xem xét kỹ hơn các thành phần được liệt kê cho Spaghetti Bolognese.

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("MATCH (REC1:Recipe{Name:'Spaghetti Bolognese'})-
    [r:Contains]->(ING:Ingredient) RETURN REC1.Name, ING.Name;")
```



Hình 7.17 Các nguyên liệu có thể có của Spaghetti Bolognese

Truy vấn Cypher chỉ liệt kê các nguyên liệu được liên kết với Spaghetti Bolognese. Hình 7.17 hiển thị kết quả trong giao diện web Neo4j.

Hãy nhớ lại nhận xét mà chúng tôi đã đưa ra khi lập chỉ mục dữ liệu trong Elaticsearch. Một tìm kiếm nhanh trên Elaticsearch về Spaghetti Bolognese cho thấy nó xuất hiện nhiều lần và tất cả các trường hợp này được sử dụng để liên kết các nguyên liệu với Spaghetti Bolognese dưới dạng công thức. Chúng ta không cần phải xem Spaghetti Bolognese như một công thức duy nhất mà hơn thế nữa là một tập hợp các cách mọi người tạo ra "Spaghetti Bolognese" của riêng họ. Điều này tạo ra một cách thú vị để xem xét dữ liệu này. Mọi người có thể tạo ra phiên bản món ăn của riêng họ với sốt cà chua, rượu vang đỏ và thịt gà hoặc thậm chí họ có thể thêm súp. Với "Spaghetti Bolognese" là một món rất dễ ăn, không có gì lạ khi nhiều người yêu thích nó.

> Câu chuyện Spaghetti Bolognese là một trò tiêu khiển thú vị nhưng không phải mục đích chúng tôi tìm đến.

Đã đến lúc giới thiệu các món ăn cho người sành ăn "Ragnar" của chúng ta.

7.3.5 Bước 5: Mô hình hóa dữ liệu

Với kiến thức về dữ liệu được làm phong phú hơn một chút, chúng ta đạt được mục tiêu của bài tập này: các đề xuất.

Đối với điều này, chúng tôi giới thiệu một người dùng mà chúng tôi gọi là “Ragnar”, đây là người thích một vài món ăn. Thông tin mới này cần được cơ sở dữ liệu đồ thị của chúng tôi hấp thụ trước khi chúng tôi có thể mong đợi nó đề xuất các món ăn mới. Do đó, bây giờ chúng ta hãy tạo nút người dùng của Ragnar với một vài tùy chọn công thức.

Liệt kê 7.6 Tạo một nút người dùng thích một số công thức nhất định trong cơ sở dữ liệu đồ thị Neo4j

```
from py2neo import Graph, Node, Relationship

graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")

UserRef = graph_db.merge_one("User","Name","Ragnar")

RecipeRef = graph_db.find_one("Recipe",property_key="Name",
    property_value="Spaghetti Bolognese")
NodesRelationship = Relationship(UserRef, "Likes", RecipeRef)
graph_db.create_unique(NodesRelationship) #Commit his like to database

graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Roasted Tomato Soup with Tiny Meatballs
    and Rice")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Moussaka")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Chipolata &amp;amp; spring onion frittata")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Meatballs In Tomato Sauce")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Macaroni cheese")))
graph_db.create_unique(Relationship(UserRef, "Likes",
    graph_db.find_one("Recipe",property_key="Name",
    property_value="Peppered Steak")))
```

Nhập mô-đun

Tạo cơ sở dữ liệu biểu đồ liên quan đến đối tượng

Tạo người dùng mới gọi là “Ragnar”

Ragnar thích Spaghetti Bolognese

Tìm công thức qua tên của mónSpaghetti Bolognese

Tạo một mối quan hệ yêu thích giữa Ragnar và món spaghetti

Lặp lại quá trình tương tự như trong các dòng ở trên nhưng đối với một số món ăn khác

Trong danh sách 7.6, người sành ăn Ragnar của chúng ta được thêm vào cơ sở dữ liệu cùng với sở thích của anh ấy đối với một số món ăn. Nếu chúng ta chọn Ragnar trong giao diện Neo4j, chúng ta sẽ có hình 7.18. Truy vấn Cypher cho điều này là

```
MATCH (U:User)-[r:Likes]->(REC:Recipe) RETURN U,REC LIMIT 25
```

Không có gì ngạc nhiên trong hình 7.18: nhiều người thích Spaghetti Bolognese, và nhà ẩm thực người Scandinavi Ragnar của chúng ta cũng vậy.

Hình 7.18 Người dùng Ragnar thích một số món ăn

Đối với công cụ đề xuất đơn giản mà chúng tôi muốn xây dựng, tất cả những gì chúng tôi phải làm là yêu cầu cơ sở dữ liệu biểu đồ cung cấp cho chúng tôi những món ăn gần nhất về nguyên liệu. Một lần nữa, đây là cách tiếp cận cơ bản đối với các hệ thống đề xuất vì nó không tính đến các yếu tố như

- Không thích một nguyên liệu hoặc một món ăn.
- Lượng thích hoặc không thích. Điểm trên 10 thay vì điểm nhị phân của thích hoặc không thích có thể tạo ra sự khác biệt.
- Lượng thành phần có trong món ăn.
- Ngưỡng để một thành phần nhất định trở nên rõ ràng trong hương vị của nó. Một số thành phần nhất định, chẳng hạn như hạt tiêu cay, sẽ có tác động lớn hơn với một liều lượng nhỏ hơn so với các thành phần khác.
- Dị ứng thực phẩm. Mặc dù điều này sẽ hoàn toàn được mô hình hóa trong việc thích hoặc không thích các món ăn với một số nguyên liệu nhất định, nhưng dị ứng thực phẩm có thể nghiêm trọng đến mức một sai lầm nhỏ có thể gây tử vong. Tránh các chất gây dị ứng nên ghi đè lên toàn bộ hệ thống khuyến nghị.
- Còn nhiều điều để bạn suy ngẫm.

Nó có thể hơi bất ngờ, nhưng một lệnh Cypher duy nhất là đủ.

```
from py2neo import Graph, Node, Relationship
graph_db = Graph("http://neo4j:neo4ja@localhost:7474/db/data/")
graph_db.cypher.execute("
    MATCH (USR1:User{Name:'Ragnar'})-[l1:Likes]->(REC1:Recipe),
            (REC1)-[c1:Contains]->(ING1:Ingredient)
        WITH ING1,REC1 MATCH (REC2:Recipe)-[c2:Contains]->(ING1:Ingredient)
        WHERE REC1 <> REC2
    RETURN REC2.Name,count(ING1) AS IngCount ORDER BY IngCount DESC LIMIT 20;")
```

Đầu tiên, tất cả các công thức nấu ăn mà Ragnar thích đều được thu thập. Sau đó, nguyên liệu của chúng được sử dụng để lấy tất cả các món ăn khác dùng chung với chúng. Sau đó, các thành phần được tính cho từng món ăn được kết nối và xếp hạng từ nhiều thành phần phổ biến đến ít thành phần. Chỉ 20 món ăn hàng đầu được giữ lại; kết quả trong bảng hình 7.19.

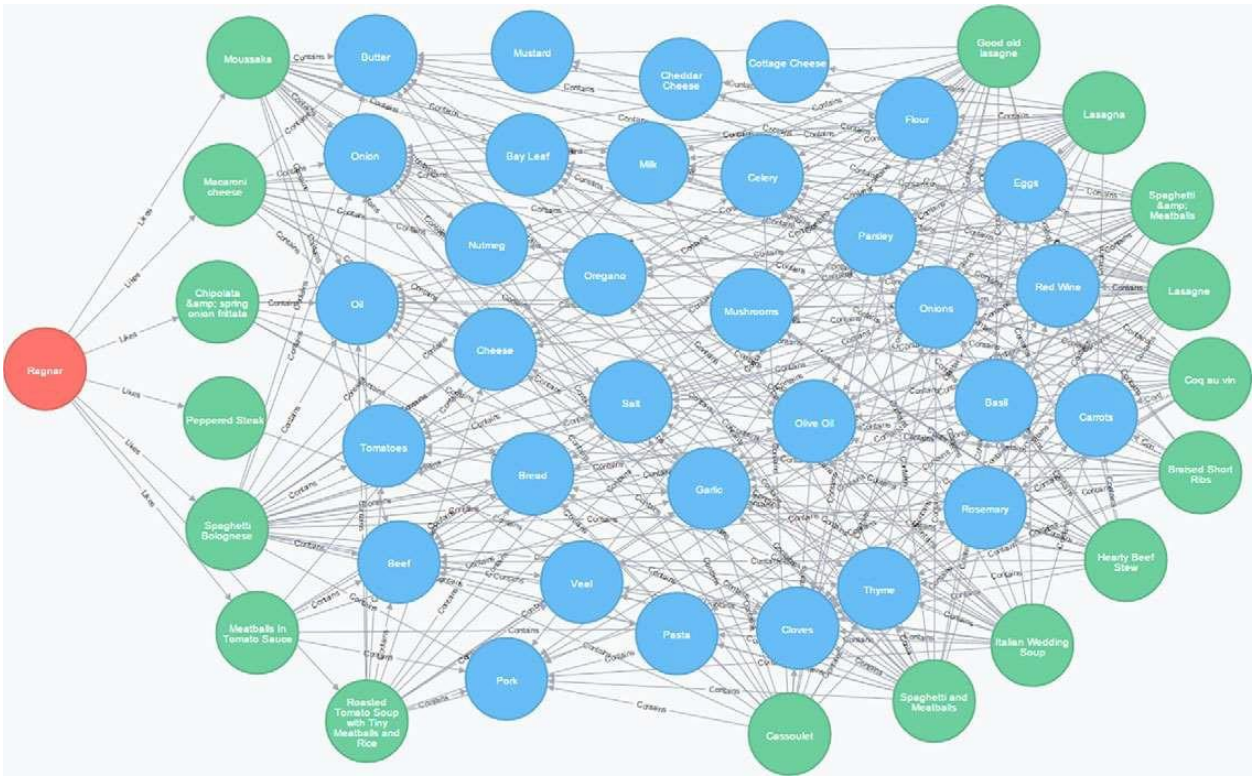
	REC2.Name	IngCount
1	Spaghetti and Meatballs	104
2	Hearty Beef Stew	91
3	Cassoulet	89
4	Lasagne	88
5	Spaghetti & Meatballs	86
6	Good old lasagne	84
7	Beef Wellington	84
8	Braised Short Ribs	83
9	Lasagna	83
10	Italian Wedding Soup	82
11	French Onion Soup	82
12	Coq au vin	82
13	Shepherd's pie	81
14	Great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

Hình 7.19 Đầu ra của đề xuất công thức; top 20 món ăn người dùng có thể yêu thích

Từ hình 7.19, chúng ta có thể suy luận rằng đã đến lúc Ragnar thử món Spaghetti và Meatballs, một món ăn đã trở nên nổi tiếng bất hủ nhờ hoạt hình của Disney Lady and the Tramp. Điều này nghe có vẻ là một đề xuất tuyệt vời cho ai đó rất thích các món ăn có mì ống và thịt viên, nhưng như chúng ta có thể thấy qua số lượng nguyên liệu, còn nhiều nguyên liệu khác hỗ trợ cho đề xuất này. Để cung cấp cho chúng tôi một gợi ý nhỏ về những gì đằng sau nó, chúng tôi có thể hiển thị các món ăn ưa thích, các đề xuất hàng đầu và một vài thành phần chồng chéo của chúng trong một hình ảnh biểu đồ tóm tắt.

7.3.6 Bước 6: Trình bày

Giao diện web Neo4j cho phép chúng tôi chạy mô hình và truy xuất một biểu đồ đẹp mắt tóm tắt một phần logic đằng sau các đề xuất. Nó cho thấy cách các món ăn được đề xuất được liên kết với các món ăn ưa thích thông qua các nguyên liệu. Điều này được thể hiện trong hình 7.20 và là kết quả cuối cùng cho nghiên cứu tình huống của chúng tôi.



Hình 7.20 Mối liên kết giữa các món ăn được người dùng ưa thích và 10 món ăn được đề xuất hàng đầu thông qua lựa chọn phụ của các thành phần chồng chéo giữa chúng

Với hình ảnh biểu đồ tuyệt đẹp này, chúng ta có thể kết thúc chương này với kiến thức rằng Ragnar có một số món ăn ngon đáng mong đợi. Đừng quên thử hệ thống đề xuất cho chính bạn bằng cách chèn các tùy chọn của riêng bạn.

7.4 Tóm tắt

Trong chương này bạn đã học

- Cơ sở dữ liệu đồ thị đặc biệt hữu ích khi gặp phải dữ liệu trong đó mối quan hệ giữa các thực thể cũng quan trọng như chính các thực thể đó. So với các cơ sở dữ liệu NoSQL khác, chúng có thể xử lý độ phức tạp lớn nhất nhưng ít dữ liệu nhất.

- Cấu trúc dữ liệu đồ thị bao gồm hai thành phần chính:
- Điểm giao - *Nodes*—Đây chính là những thực thể. Trong nghiên cứu điển hình của chúng tôi, đây là những công thức và nguyên liệu.
- Cạnh - *Edges*—Các mối quan hệ giữa các thực thể. Các mối quan hệ, giống như các nút, có thể thuộc mọi loại (ví dụ: “contains”, “likes”, “has been to”) và có thể có các thuộc tính cụ thể của riêng chúng như tên, trọng số hoặc các thước đo khác.

- Chúng tôi đã xem xét Neo4j, cơ sở dữ liệu đồ thị phổ biến nhất hiện nay. Để biết hướng dẫn về cách cài đặt, bạn có thể tham khảo phụ lục B. Chúng tôi đã xem xét việc thêm dữ liệu vào Neo4j, truy vấn nó bằng Cypher và cách truy cập giao diện web của nó.
- Cypher là ngôn ngữ truy vấn dành riêng cho cơ sở dữ liệu Neo4j và chúng tôi đã xem xét một vài ví dụ. Chúng tôi cũng đã sử dụng nó trong nghiên cứu điển hình như một phần của hệ thống gợi ý món ăn của chúng tôi.
- Trong nghiên cứu điển hình của chương, chúng ta đã sử dụng Elaticsearch để dọn sạch một bãi chứa dữ liệu công thức khổng lồ. Sau đó, chúng tôi đã chuyển đổi dữ liệu này thành cơ sở dữ liệu Neo4j với các công thức và nguyên liệu. Mục tiêu của nghiên cứu điển hình là giới thiệu các món ăn cho mọi người dựa trên sự quan tâm trước đó đối với các món ăn khác. Đối với điều này, chúng tôi đã sử dụng sự liên kết của các công thức nấu ăn thông qua các thành phần của chúng. Thư viện py2neo cho phép chúng tôi giao tiếp với máy chủ Neo4j từ Python.
- Hóa ra cơ sở dữ liệu đồ thị không chỉ hữu ích cho việc triển khai hệ thống đề xuất mà còn cho việc khám phá dữ liệu. Một trong những điều chúng tôi phát hiện ra là sự đa dạng (khôn ngoan về nguyên liệu) của các công thức nấu mì Spaghetti Bolognese ngoài kia.
- Chúng tôi đã sử dụng giao diện web Neo4j để tạo bản trình bày trực quan về cách chúng tôi nhận được từ tùy chọn món ăn đến đề xuất món ăn thông qua các nút thành phần.