Nhóm trưởng nhóm 4: Trương Thanh Sỉ - 1212327

Điện thoại: 01644530156/Email: truongthanhsi94@gmail.com

Tóm tắt nội dung công việc

Nhóm đã tìm hiểu về topic 3a(Graph store) và 3b(Neo4j) đây là hai mô hình dữ liệu NoSQL dùng để lưu trữ dữ liệu theo dạng đồ thị.

MÔN HỌC Phát triển ứng dụng Hệ thống thông tin hiện đại

BÁO CÁO SEMINAR

**THÔNG TIN NHÓM SINH VIÊN THỰC HIỆN**

|  |  |  |  |
| --- | --- | --- | --- |
| Tên nhóm: 4 | | Số lượng thành viên: 5 | |
| Hình ảnh nhóm: | | | |
| MSSV | Họ tên | Email | Điện thoại |
| 1212327 | Trương Thanh Sỉ | [Truongthanhsi94@gmail.com](mailto:Truongthanhsi94@gmail.com) | 01644530156 |
| 1212209 | Nguyễn Đức Hoàng Long | [Ndhlong1994@gmail.com](mailto:Ndhlong1994@gmail.com) | 01688653287 |
| 1212205 | Nguyễn Phượng Lĩnh | [1212205@student.hcmus.edu.vn](mailto:1212205@student.hcmus.edu.vn) | 01689357608 |
| 1212245 | Hoàng Trung Nam | [htnamitus@gmail.com](mailto:htnamitus@gmail.com) | 0967609956 |
| 1212273 | Lê Nguyễn Nhạc | [Nguyennhac94@gmail.com](mailto:Nguyennhac94@gmail.com) | 01663277533 |

**BẢNG PHÂN CÔNG CÔNG VIỆC VÀ ĐÁNH GIÁ THÀNH VIÊN NHÓM**

|  |  |  |  |
| --- | --- | --- | --- |
| Công việc thực hiện | Người thực hiện | Mức độ hoàn thành | Đánh giá của nhóm |
| Tìm hiểu mục 7.2 topic 3b | 1212205 – Nguyễn Phượng Lĩnh | 100% | 9/10 |
| Tìm hiểu mục 7.1, 7.5 topic 3b | 1212209 – Nguyễn Đức Hoàng Long | 100% | 8/10 |
| Tìm hiểu mục 7.3 topic 3b | 1212245 – Hoàng Trung Nam | 100% | 9/10 |
| Tìm hiểu mục 7.4 topic 3b | 1212273 – Lê Nguyễn Nhạc | 100% | 9/10 |
| Tìm hiểu topic 3a | 1212327 – Trương Thanh Sỉ | 100% | 9/10 |

**BÁO CÁO KẾT QUẢ SEMINAR**

# Mục Lục

[Mục Lục 3](#_Toc434911626)

[Nội Dung: 3](#_Toc434911627)

[I. Graph store 3](#_Toc434911628)

[1. Tổng quan 3](#_Toc434911629)

[2. Liên kết dữ liệu bên ngoài với chuẩn RDF (Resource Description Format) 4](#_Toc434911630)

[3. Use cases for Graph stores 5](#_Toc434911631)

[II. NEO4J 5](#_Toc434911632)

[1. Neo4j là bảng thân thiện (Mục 7.1) 6](#_Toc434911633)

[2. Graph, Groovy và CRUD( Mục 7.2) 7](#_Toc434911634)

[3. Tìm hiểu về Rest, chỉ mục, thuật toán(Mục 7.3) 24](#_Toc434911635)

[4. Distributed High Availibality (Mục 7.4 tính sẵn có cao được phân bổ ) 32](#_Toc434911636)

[5. Tổng kết (Mục 7.5) 37](#_Toc434911637)

# Nội Dung:

## Graph store

Graph store rất quan trọng trong các ứng dụng mà cần phải phân tích mối liên hệ giữa đối tượng hoặc truy cập vào tất cả các nút trong một đồ thị trong một cách đặc biệt (ví dụ như đồ thị traversal). Graph store được tối ưu để lưu trữ các nút và các cạnh trong đồ thị cho hiệu quả và cho phép truy vấn đến những đồ thị đó. Graph databases là hữu ích cho bất kỳ vấn đề kinh doanh mà có các mối quan hệ phức tạp giữa các đối tượng như: mạng xã hội, các phương pháp dựa trên các quy tắc, tạo mashups, và hệ thống đồ thị đó có thể nhanh chóng phân tích cấu trúc mạng phức tạp và tìm kiếm các mẫu trong các cấu trúc này.

 Đến cuối phần này, bạn sẽ có thể để xác định các tính năng chính của một đồ thị lưu trữ và hiểu làm thế nào mà các graph store được sử dụng để giải quyết các vấn đề kinh doanh cụ thể. Bạn sẽ trở nên quen thuộc với các thuật ngữ đồ thị như các nút, các cạnh, và thuộc tính của nó,và bạn sẽ biết về các tiêu chuẩn W3C công bố dữ liệu cho đồ thị. Bạn cũng sẽ thấy làm thế nào các graph store được thực hiện có hiệu quả tại các công ty để thực hiện liên kết phân tích, sử dụng với các phương pháp và suy luận, và tích hợp dữ liệu liên quan.

### Tổng quan

Graph store là hệ thống các nút liên quan liên kết lại với nhau tạo thành 1 đồ thị. Mỗi đồ thị gồm có 3 trường dữ liệu: nút, cạnh và thuộc tính. Các nút thường được đại diện cho các đối tượng trong thế giới thực như con người, tổ chức, sđt, … thậm chí là các tê bào trong cơ thể sống. Cạnh là thứ kết nối giữa các đối tượng lại với nhau và được biểu diễn bằng các cạnh nối giữa các nút. Graph stores liên kết các nút lại với nhau bằng cách tạo ra các định danh cho các nút và sử dụng các định danh đó để nối chúng lại với nhau thành 1 mạng lưới.

Graph store thường được sử dụng khi bạn cần lưu các dữ liệu phức tạp mà liên quan với nhau, mà mỗi liên hệ của nó đều có các thuộc tính riêng. Graph store cho phép bạn thực hiện các câu query đơn giản để show ra các nút lân cận. Tốc độ tính toán nhanh là 1 đặc tính tự nhiên của mỗi nút, nhờ vậy mà nó có khả năng giữ lại các dữ liệu của đồ thị trong RAM, là do khi 1 đồ thị được nạp vào bộ nhớ, thì nó có thể lấy lại dữ liệu mà không yêu cầu input hay output.

Graph store thì khó có thể mở rộng ra trên nhiều máy chủ. Các nút trong đồ thị đều được liên kết chặt chẽ, do đó khi mở rộng ra trên nhiều máy chủ thì sẽ phá huỷ tính chặt chẽ của đồ thị mặc dù dữ liệu có thể được nhân rộng trên nhiều máy chủ để performance được nâng cao nhưng việc ghi các nút trên nhiều server thì thực hiện phức tạp.

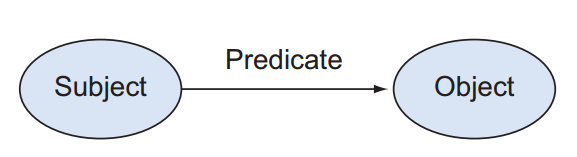
### Liên kết dữ liệu bên ngoài với chuẩn RDF (Resource Description Format)

Bạn có thể tạo thêm các phương thức riêng để xác định xem các nút nào cùng tham chiếu đến 1 điểm trong đồ thị. Một số graph store sẽ cấp ID cho mỗi nút để load những nút đó vô RAM.

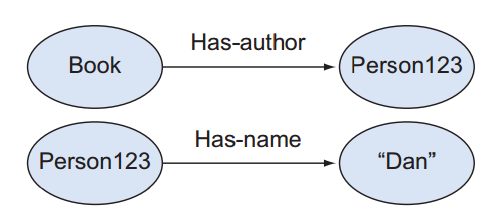
Sử dụng các URL để định danh cho các nút và lưu các thông tin về các liên kết giữa các page cũng như các nút lại với nhau sẽ được W3C tổng quát hoá lại vô 1 chuẩn gọi là RDF. RDF được tạo ra để bạn có thể tham gia vào các tập dữ liệu ngoài của các tổ chức khác nhau.

Ta có thể load cơ sở dữ liệu từ ngoài vào graph store và có thể thực hiện các câu truy vấn trên database mới – là database sau khi các tập dữ liệu bên ngoài sáp nhập vô. Đặc điểm nhận dạng là khi có nhiều nút cùng tham chiếu đến 1 đối tượng được thể hiện qua đồ thị có hướng bởi RDF, hướng từ nút source đến nút destination.

Các thuật ngữ như source, link, destination có thể thay đổi tuỳ thuộc vào tình huống sử dụng nhưng thường thì các subject, predicate, object thì được sử dụng.



Những thuật ngữ này xuất phát từ hệ thống logic và ngôn ngữ. Những thuật ngữ này để mô tả các nút được xác định trong chuẩn RDF của W3C. Trong RDF, mối quan hệ node-arc-node được gọi là triple và được kết hợp với thực tế. Hình phía dưới có 2 xác định. Thứ nhất là Book, has-author, Person123. Thứ hai là Person123, has-name, “Dan”



Khi lưu trong graph store, có 2 câu lệnh độc lập có thể được lưu trong các hệ thống khác nhau. Person123 thì đều có trong cả 2 xác định trên, do đó mà ứng dụng của mình có thể nhận ra rằng Author của Book có tên là “Dan”.



Trên thực tế thì hai nút ở 2 nhóm khác nhau có thể tham chiếu đến cùng 1 đối tượng vật lý. Ví dụ, nút Person123 ở 2 đồ thị trên là như nhau. Do đó, ta có thể ghép chúng lại thành 1 đồ thị như phía dưới. Quá trình này rất hữu ích trong các mô hình phức tạp cũng như trong các lĩnh vực đòi hỏi sự suy luận.

Có 1 phương pháp có thể xác định HTML bất cứ ở đâu trên thế giới bằng cách thống nhất cấu trúc tài nguyên sử dụng, do đó ta có thể sử dụng cấu trúc này ở bất kì khi nào có thể. Khác với URL, URI không trỏ đến bất kì trang web nào. Do đó, bạn phải tìm ra cách làm cho chúng nhất quán trên toàn bộ trang web và phù hợp khi so sánh 2 nút với nhau.

Trong thực tế, mỗi triple thường liên kết thông tin với các triple khác. Các thuộc tính thường được sử dụng để liên kết các triple lại với nhau được gọi là siêu dữ liệu liên kết vì chúng mô tả thông tin về sự liên kết đó. Việc lưu trữ siêu dữ liệu liên kết thì không mất quá nhiều không gian đĩa cứng nhưng nó lại giúp ta dễ dàng kiểm soát và quản lý dữ liệu hơn.

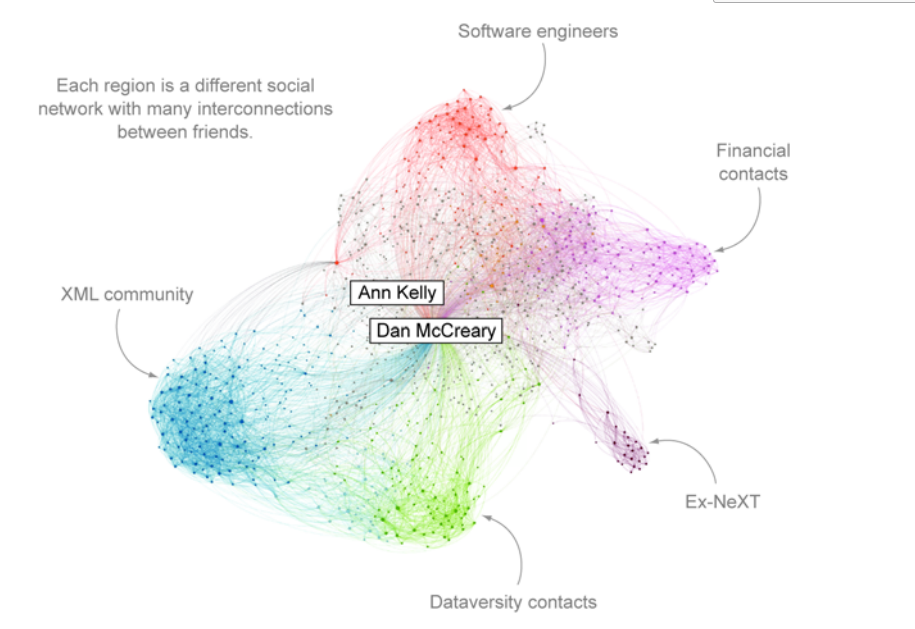
### Use cases for Graph stores

Các tình huống mà graph store có thể được sử dụng hiệu quả để giải quyết 1 vấn đề cụ thể:

* Link analysis được sử dụng khi ta muốn thực hiện tìm kiếm các mô hình và các mối quan hệ trong các tình huống sử dụng như mạng xã hội, điện thoại, email.
* Rules and inference được sử dụng khi muốn truy vấn đến các cấu trúc phức tạp
* Integrating linked data được sử dụng với số lượng lớn các siêu dữ liệu mở để thực hiện các công việc trong thời gian thực và xây dựng các mashup mà không cần phải lưu dữ liệu.

#### 3.1 LINK ANALYSIS

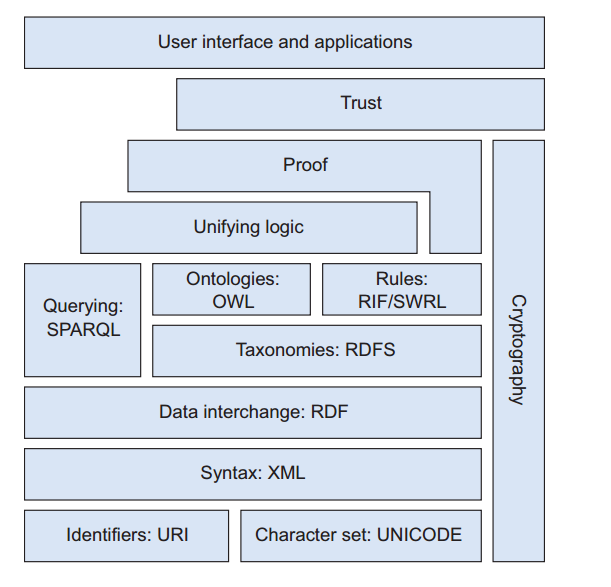
* Cách tốt nhất để giải quyết 1 vấn đề kinh doanh đó là thể hiện vấn đề này lên graph store. Hình dưới là ví dụ về đồ thị về mạng xã hội. VD: Khi mình thêm 1 bạn mới vào list frend của mình, mình cũng muốn biết xem người bạn mình dự định sẽ thêm vào có ai là bạn chung với mình không? Và để có được thông tin này, trước tiên mình sẽ cần cái list friend của mình . Và mỗi người trong list friend của mình thì đều có 1 list friend riêng của họ. Và nếu như trong list friend của họ có tên của bạn thì đó là những người bạn chung.



* Graph store có thể thực thi nhiều hoạt động trong quá trình phân tích nhanh hơn bằng cách sử dụng kỹ thuật loại bỏ các nút không cần từ bộ nhớ. Mặc dù vậy, nhưng cũng cần phải đủ RAM để có thể lưu trữ tất cả liên kết trong quá trình phân tích.
* Graph store thường được sử dụng cho những thứ như mạng xã hội vì nó thích hợp để xác định mô hình khác biệt của các liên kết giữa các nút.
* Graph store cũng rất hữu ích trong việc liên kết hợp dữ liệu và tìm kiếm các mô hình trong các tài liệu lớn. Khai thác các thực thể là quá trình xác định các thực thể quan trọng nhất trong 1 tài liệu. Các đối tượng thường là 1 danh từ trong tài liệu như con người, ngày tháng, nơi chốn, sản phẩm. Khi các thực thể chính đã được xác định, chúng được sử dụng để thực hiện các chức năng tìm kiếm nâng cao.
* Quá trình khai thác thực thể (là 1 loại của “Xử lý ngôn ngữ tự nhiên” (NLP)) có thể kết hợp với các công cụ khác để trích xuất dữ kiện đơn giản. Ví dụ cho câu sau “John Adams was born on October 19, 1735” có thể được chia thành các xác định sau:
* Hồ sơ về 1 người là “John Adams” và là 1 chủ đề
* Một mối liên hệ liên kết subject với object.
* Đối tượng về ngày có gái trị là “October 19, 1735”
* Mặc dù xác định đơn giản có thể dễ dàng tìm thấy bằng cách xử lý NLP, nhưng quá trình hiểu được từng câu hoàn toàn có thể trở nên phức tạp và phụ thuộc vào ngữ cảnh sử dụng, tình huống sử dụng. Các xác định này tốt nhất nên được thể hiện trong cấu trúc đồ thị

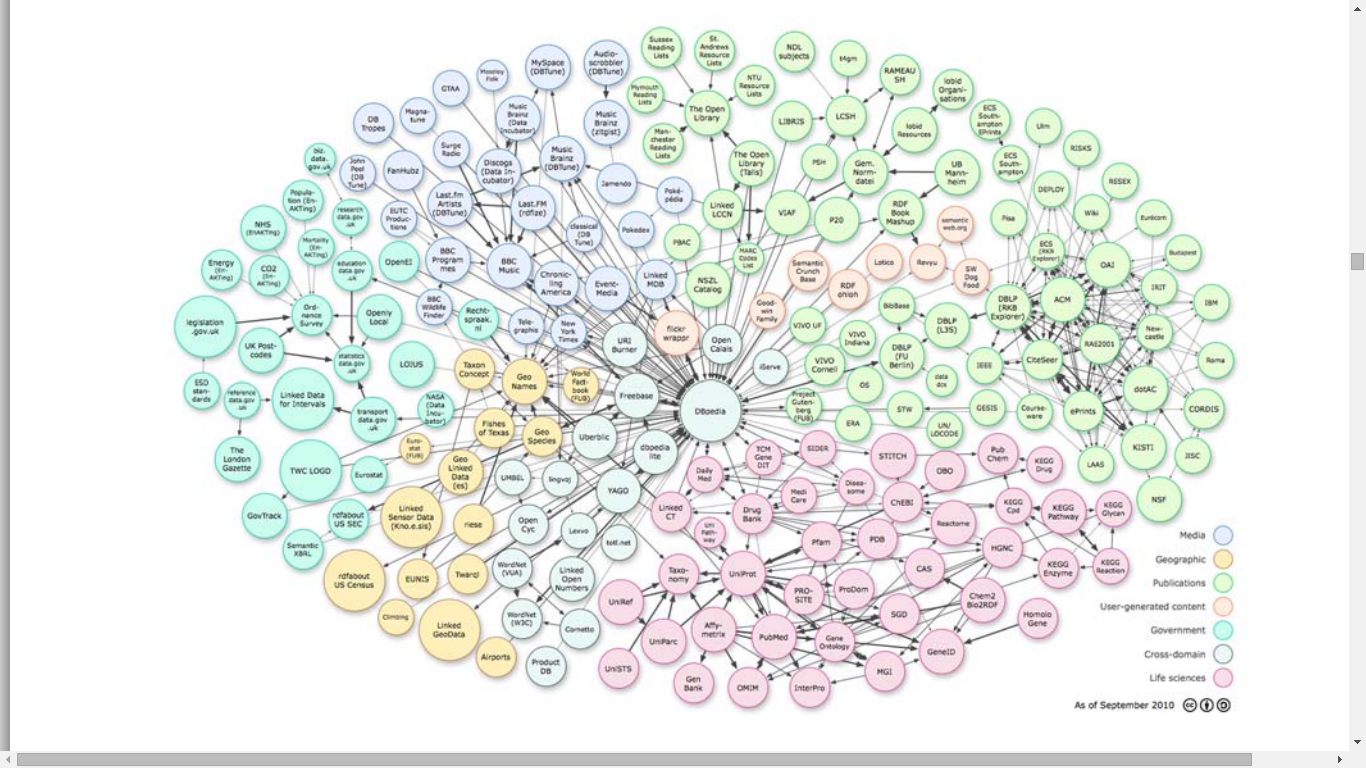
#### 3.2 GRAPHS, RULES, AND INFERENCE

* Việc quy định các thuật ngữ thì phụ thuộc vào bối cảnh, tình huống, địa điểm sử dụng. Ở đây, các thuật ngữ này được sử dụng để xác định các quy tắc trừu tượng liên quan đến sự hiểu biết các đối tượng trong hệ thống và làm thế nào các thuộc tính của đối tượng cho phép ta hiểu được đối tượng sâu hơn, và sử dụng tốt các tập dữ liệu lớn.
* RDF được thiết kế để trở thành 1 chuẩn trong nhiều loại vấn đề trong cấu trúc đồ thị. Mục đích chính của việc sử dụng RDF là để lưu trữ các logic và luật. Và khi đã thiết lập các luật này, ta có thể suy luận hoặc sử dụng luật để khám phá về 1 điều khác trong hệ thống. Việc sử dụng RDF và suy luận không giới hạn đến mạng xã hội và đánh giá sản phẩm. RDF là 1 cấu trúc với mục đích chung mà có thể được sử dụng để lưu trữ nhiều hình thức logic
* Trong phần phân tích về các liên kết thì ta đã hiểu cách văn bản được mã hoá với các thực thể như con người, ngày tháng để giúp ta tìm thấy các hoạt động liên quan. Bây giờ, ta có thể làm những thứ to tác hơn để có được các thông tin bổ sung từ các hoạt động để giải quyết 1 vấn đề kinh doanh.
* W3C là 1 framework của chuẩn sử dụng RDF để giải quyết các vấn đề kinh doanh. Framework này thường được gọi là Semantic Web Stack. Hình phía dưới là 1 loại Semantic Web Stack với các chuẩn cơ bản như URI, XML, RDF ở dưới cùng của ngăn xếp. Ở giữa thì gồm các chuẩn truy vấn (SPARQL), chuẩn quy tắc (RIF/ SWRL). Ở trên cùng của ngăn xếp là giao diện người dùng và ứng dụng trên lớp trừu tượng logic, proof, trust. Các lớp trừu tượng này vẫn chưa được chuẩn hoá. Đây cũng là nơi tập trung nhiều nghiên cứu và phát triển Semantic Web. Còn những phần đã được chuẩn hoá thì có thể an toàn trong việc trao đổi dữ liệu trên internet.



#### 3.3 USING GRAPHS TO PROCESS PUBLIC DATASETS

* Graph store thì rất hữu ích trong việc phân tích các dữ liệu khi các dữ liệu đó không hoàn toàn do ta tạo ra. Điều gì xảy ra nếu ta cần phân tích 3 dataset khác nhau được tạo ra bởi ba tổ chức khác nhau? Thậm chí, các tổ chức này còn có không biết đối phương có tồn tại hay không? Vậy làm thế nào ta có thể tự động sáp nhập các dataset của họ lại với nhau để ta có thể lấy được thông tin ta cần? Làm thế nào để tạo mashup và recombinations của dữ liệu này một cách hiệu quả? Một câu trả lời duy nhất là sử dụng một bộ công cụ được gọi là Linked Open Data (LOD). Đây là 1 kỹ thuật cho phép sáp nhập các dataset khác nhau để tạo ra các ứng dụng mới và dữ liệu mới.
* Chiến lược LOD thì rất quan trọng cho bất cứ ai làm nghiên cứu hoặc phân tích các dataset có sẵn. Các nghiên cứu này có các chủ đề như khách hàng mục tiêu, phân tích xu hướng, phân tích tâm lý hoặc tạo ra các dịch vụ thông tin mới. Hợp dữ liệu vào các loại hình mới nhằm cung cấp các cơ hội cho các doanh nghiệp mới.
* LOD tạo ra các tập dữ liệu mới bằng cách kết hợp các thông tin từ nhiều dataset có sẵn cho phù hợp với cấu trúc LOD như RDF, URI. Hình dáng của các trang web LOD phổ biến được gọi là sơ đồ cloud LOD được thể hiện như hình phía dưới. Tại trung tâm của sơ đồ cloud LOD, ta sẽ thấy các trang web có chứa số lượng lớn các dataset có cùng mục đích chung.



## NEO4J

Tổng quan:

Neo4J dùng sự gắn kết giữa các cột trong bảng với nhau. Các standard database khác dựa vào sự gắn kết của cột.

Neo4J là một loại của Graph database. Dữ liệu được lưu trữ như đồ thị, được biết đến là “whiteboard friendly” – bạn có thể vẽ các thiết kế như các hộp và các dòng trên whiteboard. Neo4J tập trung vào *relationships* giữa các giá trị hơn là sự tương đồng giữa các tập hợp của giá trị (như tập hợp của các tài liệu hoặc bảng của các dòng). Theo cách này, Neo4J có thể lưu trữ tốt các biến dữ liệu một cách tự nhiên và không phức tạp.

Neo4J đủ nhỏ để nhúng vào một vài ứng dụng. Giới hạn của Neo4J có thể lưu trữ 4 tỉ đỉnh và cạnh. Các cluster của Neo4J hỗ trợ điều khiển sự tái tạo dữ liệu thông qua nhiều server, xử lý các vấn đề về kích thước

### Neo4j là bảng thân thiện (Mục 7.1)

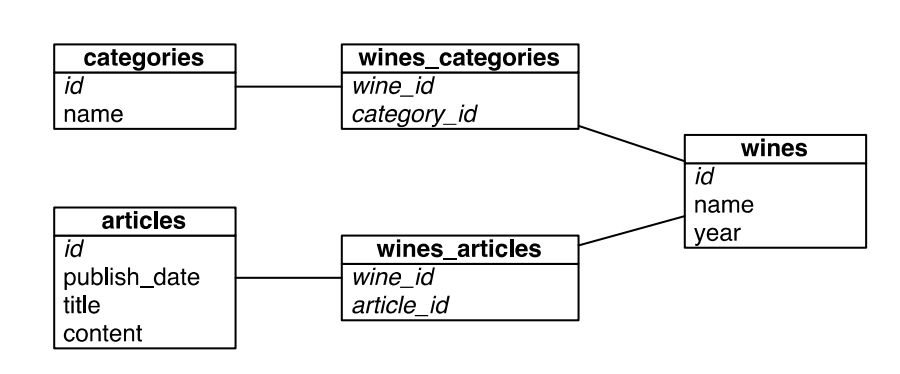
Tưởng tượng bạn phải tạo ra được lời đề nghị kỹ thuật rượu với nhiều thứ khác nhau như: Regions, wineries, vintages và designations.

1 số từ trong ý nghĩa về rượu dịch ra chỉ có tính tương đối:

* Regions: Vùng miền sản xuất.
* Vineries: Nhà máy sản xuất rượu.
* Vintages: Nơi sản xuất nho.
* Designations: Tên của rượu.

Có lẽ bạn cần lưu trữ các công thức này từ các tác giả sau khi nhận được sự miêu tả của họ về cách nấu rượu và muốn người sử dụng tìm được loại rượu yêu thích.

Bên dưới là một cách lưu trữ thông tin: Các bảng được tạo ra và liên kết với nhau qua nhiều mối quan hệ. Nhìn vào sơ đồ bên dưới không hoàn toàn đáp ứng được mong muốn của bạn. Không thấy được rõ sự liên quan giữa các giá trị, rượu này được sản xuất ở vùng nào, miền nào, loại nho nào…



H1. Wine suggestion schema in relational UML

**Một câu nói từ xưa về relational database: *On a long enough timeline, all fileds become optional.* Có nghĩa là timeline quá dài thì tất cả các trường trở thành sự lựa chọn. ?????( chưa rõ)**

Neo4J hỗ trợ đầy đủ giá trị và cấu trúc chỉ những cái cần thiết. Như lược đồ bên dưới nhìn vào ta có thể dễ dàng thấy được các thông tin và cấu trúc cần thiết: năm sản xuất 2007, rượu với tên là prancing wolf, loại: riesling (một loại rượu nho nước Đức), thông tin dựa trên wine expert(chuyên gia thẩm định rượu).



H2. Wine suggestion schema in relational UML

### Graph, Groovy và CRUD( Mục 7.2)

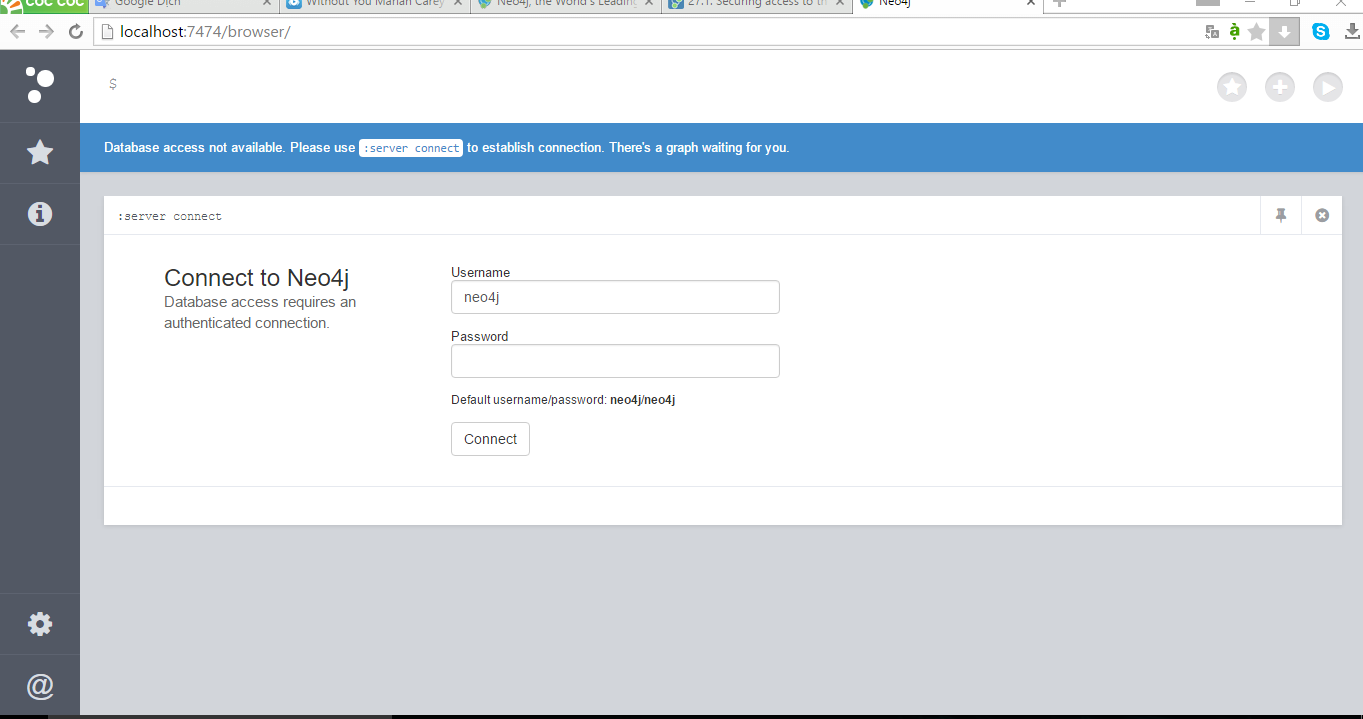
Trước hết bắt đầu với web interface của Neo4J để tìm hiểu làm thế nào Neo4J biểu diễn diễn dữ liệu dưới dạng đồ thị và làm thế nào đi đến, truy vấn đỉnh, cạnh xung quanh đồ thị.

Hướng dẫn cài đặt:

Truy cập <http://neo4j.com/> download và giải nén Neo4j package, cd đến thư mục khởi động server:

$ bin/neo4j start

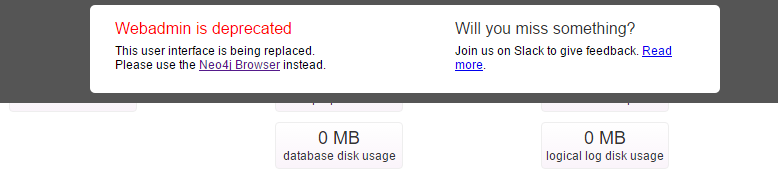
Sau đó chạy: <http://localhost:7474/browser/>



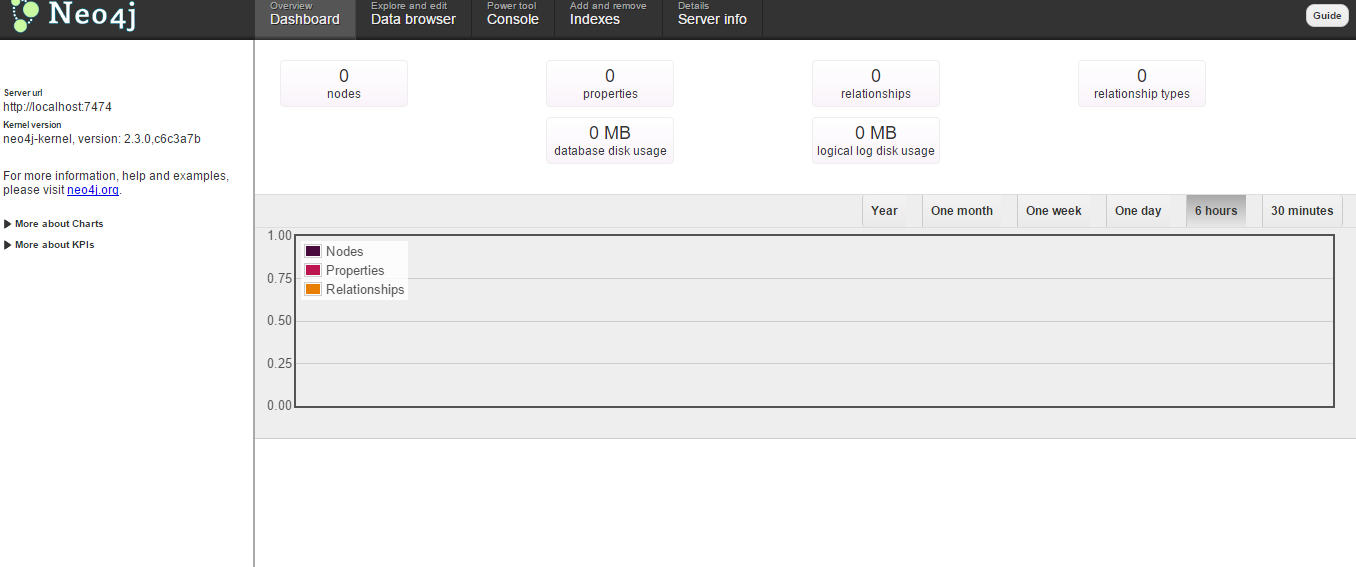
Nhập username và password. Bắt đầu tiến hành sử dụng

#### 2.1 Neo4j’s Web Interface

Chạy: <http://localhost:7474/webadmin/>

Theo tài liệu hướng dẫn thì sử dụng Neo4j 1.7. Phiên bản này khá cũ nên so với bản mới Neo4J 2.3 sẽ hiện cảnh báo:  


Tuy nhiên vẫn hỗ trợ webadmin



Phần giao diện Dashboard sẽ cho xem tổng quan về số node, số relationship, số property, relationship type… 1 vài thông tin của đồ thị. Ban đầu thì các giá trị này sẽ là 0 vì chưa tạo các node và quan hệ.



Các node, relationship còn được gọi là vertex và edges. Mỗi node và relationship đều bao gồm property và value.

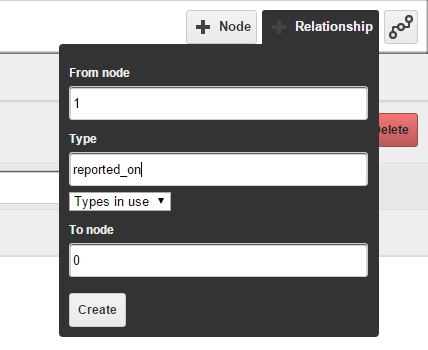
Lấy ví dụ về Wine đề cập ở phần trên.

Ta sẽ thêm một node với property là name và value là Prancing Wolf Ice Wine 2007 để biểu diễn về loại rượu cụ thể và năm sản xuất. Click vào +Node để thêm 1 node:



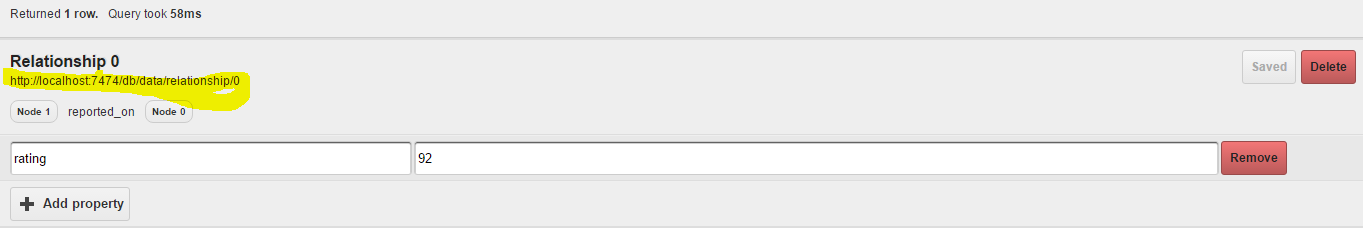
Tương tự ta cũng thêm một node với property là name và value là Wine Expert Monthly (có thể viết ngắn gọn như [name: “Wine Expert Monthly”]). Chú ý khi thêm thì chỉ số các node sẽ tự tăng (0, 1, 2, 3…n)

Bây giờ ta đã có 2 node, từ 2 node Wine Expert Monthly và Pracing Wolf wine, cần thêm mối quan hệ giữa 2 node này.Click + Relationship và thiết lập từ 1 tới 0 với type là reported\_on



Để xem relationship truy cập đường link: <http://localhost:7474/db/data/relationship/0>

Cũng như node thì relationship cũng có property với value. Ta add property [rating:92] để đánh giá điểm rượu nhận được.

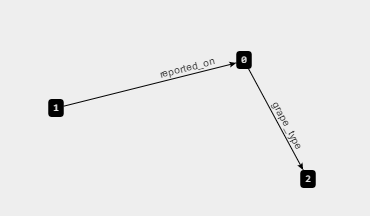


Sau khi biết loại rượu được tạo ra từ ai ta sẽ thêm một node để xác định loại nho nào.

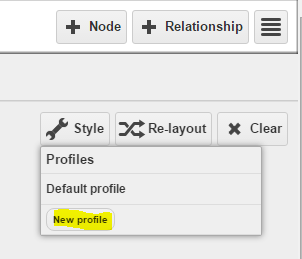
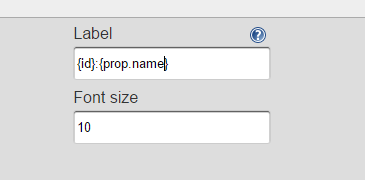
Add node riesling grape (nho reisling). Thêm property [name: “riesling”]. Sau đó tạo relationship từ 0 tới 2 với type là grape\_type và property [style: “ice wine”].

Để xem đồ thị ta click vào button ngoài cùng bên phải: 

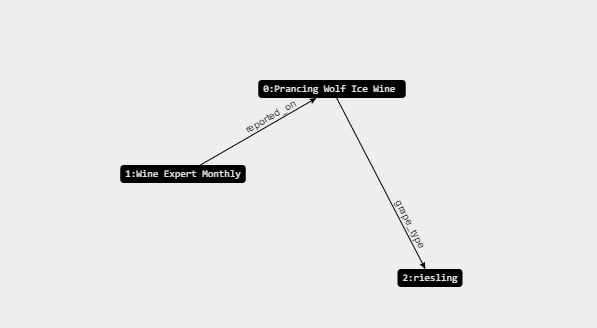
Được đồ thị như sau:



Nhìn vào đồ thị ta sẽ thấy được tên các relationship nhưng thật khó để biết node 0, 1, 2 là gì. Để trực quan ta sẽ điều chỉnh nhãn cho các node. Vào chọn new profile

Điểu chỉnh label từ {id} sang {id}:{prop.name}. Nhấn save. Ta được kết quả như sau

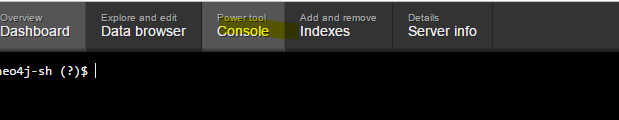


Với giao diện web interface dễ dàng thao tác thêm, xóa, sửa một node, relationship hỗ trợ cho việc làm việc với graph.

#### 2.2 Neo4j thông qua Gremlin

Có rất nhiều ngôn ngữ tương thích với Neo4j như: Java code, REST, Cypher, Ruby console và nhiều ngôn ngữ khác. Ở tài liệu này sẽ sử dụng một ngôn ngữ là Gremlin được viết dựa trên ngôn ngữ lập trình Groovy. Bạn không cần biết về Groovy để sử dụng Gremlin, nó cũng tương tự như SQL.

Gremlin hỗ trợ các thao tác cơ bản, bạn có thể sử dụng Groovy và java libraries trong Gremlin. Gremlin có sẵn trong giao diện web admin.



Quy ước cụ thể trong Gremlin, g là biến đại diện cho graph object. Các Graph action function sẽ gọi đến nó.

Trong Neo4j gọi mỗi điểm trong graph data là node, gremlin sẽ gọi là vertex và relationshop, gremlin gọi là edge.

Để truy xuất tất cả các đỉnh(vertex) trong đồ thị ta dùng thuộc tính V

gremlin> g.V

==> v[0]

==> v[1]

==> v[2]

Tương tự với cạnh ta cũng có thuộc tính E để truy cập tất cả các cạnh(edge)

gremlin> g.E

==> e[0][1-reported\_on->0]

==> e[1][0-grape\_type->2]

Để lấy 1 đỉnh cụ thể bạn có thể thông qua node number của v method

Gremlin> g.v(0)

==> v[0]

Để đảm bảo chính xác vertex, bạn có thể truy xuất thông qua property của đỉnh thông qua map() metho

Gremlin> g.v(0).map()

==> name=Prancing Wolf Ice Wine 2007

Mặc dù có thể sử dụng v(0) để nhận chính xác node, bạn cũng có thể lọc các node thông qua value nếu mốn. Ví dụ, bạn có thể nhận reiesling từ name, với cú pháp như bên dưới:

Gremlin> g.V.filter{it.name==’riesling’}

==> v[2]

Khi bạn có một đỉnh (vertex), bạn muốn lấy đầu ra của cạnh(edge) là một đỉnh(vertex) khác thông qua method outE().

Gremlin> g.V.filter {it.name==’Wine Expert Monthly’}. outE()

==> e [0][1-reported\_on->0]

Từ một cạnh out, bạn có thể trả về value chính nó bằng inV. Reported\_on từ Wine Expert đến Prancing Wolf Ice Wine 2007, outE.inV sẽ trả về chính nó. Dùng property name để lấy giá trị

Gremlin>g.v.filter {it.name==’Wine Expert Monthly’}.outE.inV.name

==> Prancing Wolf Ice Wine 2007

Để thêm một đỉnh(vertex) Prancing Wolf Winery và cạnh(Edge):

Gremlin> pwolf = g.addVertex([name:’Prancing Wolf Winery’])

==> v[3]

Gremlin> g.addEdge(pwolf, g.v(0), ‘produced’)

==> e[2][3-produced->0]

Tương tự ta cũng thêm 2 reiesling: Kabinett và Spatlese

Gremlin> kabinett = g.addVertex([name: ‘Prancing Wolf Kabinett 2007’])

==> v[4]

Gremlin> g.addEdge(pwolf, kabinett, ‘produced’)

==> v[3][3-produced->4]

Gremlin>spatlese = g.addVertex([name:’Prancing Wolf Spatlese 2007’])

==> v[5]

Gremlin> g.addEdge(pwolf, spatlese, ‘produced’)

==> e[4][4-produced->5]

Bắt đầu wrap up graph thêm một vài cạnh từ riesling. Thiết lập biến riesling lọc theo riesling node, method next() là cần thiết để lấy đỉnh đầu tiên trong đường.

Gremlin> riesling = g.V.filter{it.name==’riesling’}.next()

==> v[2]

Gremlin> g.addEdge([style:’kabinett’], kabinett, riesling, ‘grape\_type’)

==> e[5][4-grape\_type->2]

Kết quả sau thu được:



#### 2.3 The Power of Pipes

Bạn có thể nghĩ Gremlin hoạt động như series của các pipe. Mỗi pipe sẽ chọn input và đưa ra các tập như output. Mỗi tập có thể có một item, nhiều item hoặc không item. Mỗi item có thể là vertices, edges hoặc property value.

Ví dụ, outE pipe đặt một tập hợp các đỉnh và gửi đi tập hợp các cạnh. Series của các pipe gọi là pieline và expresses declaratively (các thể hiện khai báo) điều này là vấn đề. Trái ngược với hướng tiếp cận lập trình điển hình, điều này yêu cầu bạn phải mô tả các bước để giải quyết vấn đề. Sử dụng pipe là một cách ngắn gọn để truy vấn graph database.

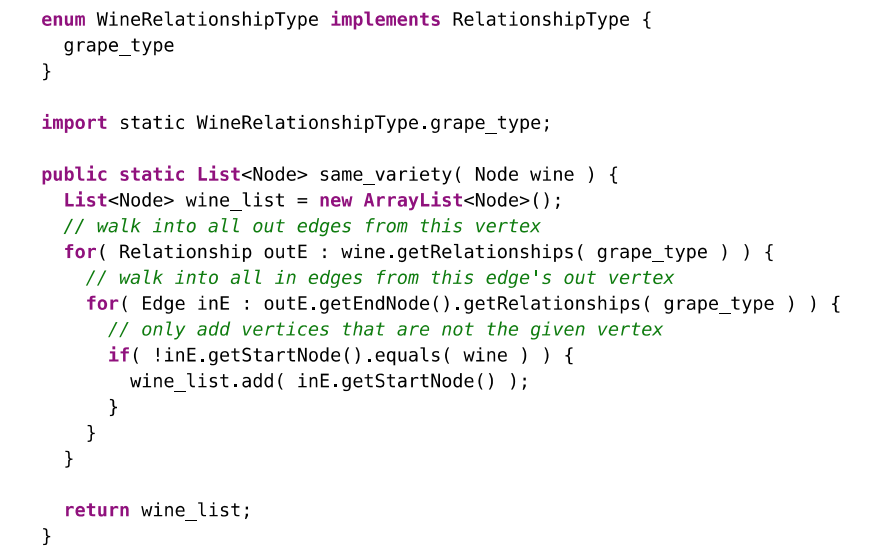
Gremlin là một ngôn ngữ xây dựng nhiều pipe. Đặc biệt nó xây dựng Java project tên Pipes. Để tìm hiểu khái niệm pipe, chúng ta trở lại wine graph. Giả sử bạn muốn tìm wines giống với rượu nhận được, chúng ta có các loại giống nhau. Bạn có theo dõi ice wine cũng có thể chia sẻ cạnh grape\_type với các node khác.

Ice\_wine = g.v(0)

Ice\_wine.out(‘grape\_type’).in(‘grape\_type’).filter{!it.equals(ice\_wine)}

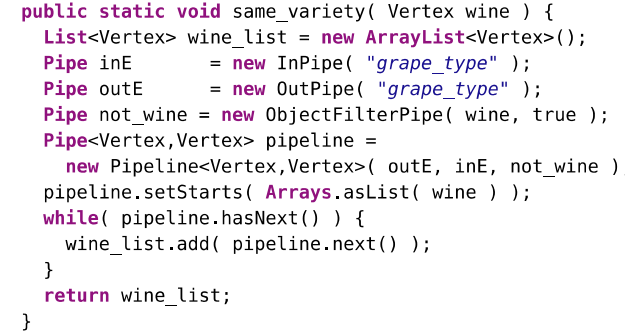
Từ đầu ra của ice\_wine là riesling ta sẽ tìm các đầu vào in có cạnh grape\_type trừ ice\_wine

Nếu bạn làm việc với Samlltalk hoặc Rails, mỗi kiểu method dường như tương tự. Nhưng so sánh trước khi sử dụng chuẩn Neo4j Java API trình bày tiếp theo, ở đây các quan hệ node phải tích hợp thông qua việc truy cập các node khác nhau.



Thay vì trộn và tích hợp như một thể hiện sớm. Project Pipes thiết kế một cách để khai báo các đỉnh vào và ra. Bạn có thể tạo biểu đồ của in và out pipe, theo dấu và yêu cầu values từ pipeline. Sau khi lặp đi lặp lại method hasNext() của pipeline, nó sẽ trả về node tiếp theo phù hợp. Trong cách khác, pipeline đi qua cây cho bạn. Cho đến khi pipeline yêu cầu, bạn khai báo cơ bản làm thế nào cách đi đó xảy ra.

Như minh họa, ở đây thực hiện method same\_variety(). Sử dụng Pipes hơn một cách rõ ràng vòng lặp:



Sâu bên dưới Gremlin là Pipe-building language. Công việc đi qua đồ thị vẫn hoàn tất trên Neo4j server, nhưng Gremlin cơ bản cố gắng xây dựng truy vấn Neo4j có thể hiểu được.

#### 2.4 Pipeline vs. Vertex

Để lấy một tập hợp chưa chỉ một đỉnh đặc biệt, bạn có thể theo vết nó từ một dãy các node. Lấy ví dụ, g.V.filter {it.name == ‘reisling’}. V property là danh sách các node, từ điều này chúng ta chọn lọc danh sách phụ. Nhưng khi bạn muốn đỉnh chính nó, bạn cần gọi method next(). Method nhận đỉnh đầu tiên từ pipeline. Nó giống với sự khác nhau giữa mảng của một thành phần và chính nó.

Nếu bạn tìm kiếm class constructed từ việc gọi property của filter class, chú ý nó trả về GremlinPipeline

Gremlin> g.V.filter{it.name==’Prancing Wolf Winery’}.class

==> class com.tinkerpop.gremlin.pipes.GremlinPipeline

So sánh với class của node kế tiếp từ pipeline. Nó trả về điều ngược lại, Neo4jVertex

Gremlin> g.V.filter{it.name==’Prancing Wolf Winery’}.next().class

==> class com.tinkerpop.blueprints.pgm.impls.neo4j.Neo4jVertex

Mặc dù các console thuận tiện đưa ra danh sách node nhận được từ pipeline, nó cũng giữ pipeline đến khi bạn nhận được vài thứ từ nó.

#### 2.5 Schemaless Social

Tạo một khái cạnh xã hội từ graph dễ để thêm nhiều node. Giả sử bạn muốn thêm 3 người-2 người biết lần nhau và một xa lạ, mỗi người thích một loại rượu của mình.

Alice một chút ngọt và fan lớn của ice wine

Alice = g.addVertex([name:’Alice’])

Ice\_wine = g.V.filter{it.name==’Prancing Wolf Ice Wine 2007’}.next()

g.addEdge(alice, ice\_wine, ‘likes’)

Tom thích Kabinett và ice wine và tin những thứ từ Wine Expert Monthly

Tom = g.addVertex([name:’Tom’])

Kabinett = g.V.filter{it.name==’Prancing Wolf Kabinett 2002’}.next()

g.addEdge(tom, kabinett,’likes’)

g.addEdge(tom, ice\_wine, ‘likes’)

g.addEdge(tom,g.V.filter{it.name=’Wine Expert Monthly’}).next(), ‘trusts’)

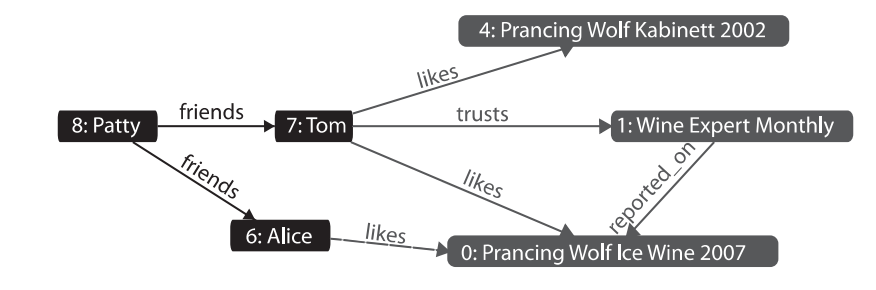
Patty là bạn của Tom và Alice nhưng là người mới về rượi, có một vài lựa chọn yêu thích.

Patty = g.addVertex([name:’Patty’])

g.addEdge(patty, tom, ‘friends’)

g.addEdge(patty, alice, ‘friends’)

Không có thay đổi căn bản cấu trúc của đồ thị tồn tại, chúng ta có thể thêm vào thể hiện bên ngoài ý định ban đầu. Những node mới liên quan, thể hiện như bên dưới:



#### 2.6 Stepping Stones

Chúng ta tìm kiếm một vài các step của Gremlin hoặc Các đơn vị của quá trình Pipe. Gremlin cung cấp nhiều hơn những điều này. Chúng ta hãy tiếm kiếm nhiều hơn việc xây dựng ác block (khối) không chỉ đi đến graph mà còn chuyển đổi đối tượng, thấu dấu các bước và thực thi nhiều mặt tác động như việc tính toán các nhóm node theo tiêu chí.

Chúng ta thấy inE, outE, inV và outV, điều này chuyển đổi qua các bước nhận được điểm đến và điểm đi cảu các edge (cạnh) và vertices (các cạnh). Hai loại khác là bothE và bothV chỉ theo dõi các edge (cạnh, bất kể cho dù in và out trực tiếp hay không.

Việc lấy cả Alice và tất cả bạn của cô ấy. Chúng ta đặt name đến cuối để lấy mỗi điểm name property. Vì chúng ta không chú ý chiều hướng cạnh friend, chúng ta sử dụng bothE và bothV.

Alice.bothE(‘friends’).bothV.name

==> Alice

==> Patty

Nếu bạn không muốn Alice, method except() sẽ giúp bạn bỏ qua danh sách node bạn không muốn và đi đến node còn lại.

Alice.bothE(‘friends’).bothV.except([alice]).name

==> Patty

Ngược lại cảu except() là retain(). Bạn có thể đoán, đi đến chỉ những node phù hợp.

Một lựa chọn khác để theo dấu từ đỉnh cuối với code block, Ở đây các bước hiện tại không so sánh với đỉnh alice

Alice.bothE(‘friends’).bothV.filter{!it.equal(alice)}.anme

Nếu bạn muốn biết bạn của bạn Alice. Bạn phải lặp lại bước như sau:

Alice.bothE(‘friends’).bothV.except([alice]).

bothE(‘friends’).bothV.except([alice])

Giống với cách trên, bạn muốn lấy những người bạn của bạn Alice từ việc thêm nhiều bothE/bothV/except để lấy được chuỗi cần thiết. Nhưng điều này quá dài dòng và nó không có thể được quản lý khi viết số lượng biến của nhiều bước. Mothod loop() sẽ làm điều này. Nó lặp lại một vài số của các bước trước và tiếp tục cho đến khi nhận kết quả đúng.

Bên dưới là code sẽ lặp trước 3 bước từ việc đếm giai đoạn qua lại gọi loop. Except là một, bothV là 2 và bothE là 3.

Alice.bothE(‘friends’).bothV.except([alice]).loop(3){

It.lopps <= 2

}.name

Tại mỗi thời điểm thông qua các bước của vòng lặp, loop() gọi ra các lời gọi đóng kín, đoạn mã giữa {…}. Tại đây, thuộc tính it.loops giữ theo dõi bao nhiêu thời gian vòng lặp cho đến khi thực thi. Trong trường hợp của chúng ta, chúng ta sẽ kiểm tra và trả về số lượng nhỏ hơn bằng 2, nghĩa là vòng lặp thực hiện 2 lần và dừng. Trong tầm ảnh hưởng, việc đóng kín là rất giống mệnh đề while loop trong ngôn ngữ lập trình.

==> Tom

==> Patty

==> Patty

Vòng lặp làm việc, chính xác tìm thấy Tom và Patty. Nhưng chúng ta bị lặp 2 Patty. Bởi vì mỗi Patty là bạn của Alice, và có nhiều khớp bởi vì cô ấy là bạn với Tom. Sử dụng method deup() sẽ không bị trùng lắp.

Alice.bothE(‘friends’).bothV.except([alice]).loop(3){

It.lopps <= 2

}.deup.name

==> Tom

==> Patty

Để có cái nhìn sâu sắc bên trong đường dấn đến các giá trị, bạn có thể theo dõi friend->friend path sử dụng paths().

Alice.bothE(‘friends’).bothV.except([alice]).loop(3){

It.lopps <= 2

}.deup.name.paths

==> [v[7], e[12][9-friends->7], v[9], e[11][9-friends->8], v[8], Tom]

==> [v[7], e[12][9-friends->7], v[9], e[1][9->friends->8], v[9], Patty]

Tất cả các đường bạn có tiến xa phía trước thông qua graph. Bạn cần 2 bước phái trước và 2 bước phái sau. Bắt đầu với Alice node, bạn có thể đi đến 2 bước và quay trở lại, trả về Alice node

Gremlin> alice.outE.inV.back(2).name

==> Alice

Một bước cuối thường sử dụng chúng ta thường truy tìm là groupCount(), đi qua các node và đếm các giá trị lặp lại, đưa chúng vào map.

Cân nhắc ví dụ với property là năm của các đỉnh trong đồ thị và đếm có bao nhiêu:

Gremlin> name\_map = [:]

Gremlin> g.V.name.groupCount(name\_map)

Gremlin>name\_map

==> Prancing Wolf Ice Wine 2007 = 1

==> Wine Expert Monthly = 1

==> riesling = 1

==> Prancing Wolf Winery = 1

==> Prancing Wolf Kabinett 2002 = 1

==> Prancing Wolf Spatlese 2007 = `

==> Alice = 1

==> Tom = 1

==> Patty = 1

Trong Groovy/Gremlin, mỗi map là ký hiệu [:] và khá nhiều giống nhau trong Ruby/JavaScript {}. Chú ý tất cả giá trị là 1. Điều này chính xác là chúng tôi mong đợi, bởi vì không có sự lặp lại tên và tập V chỉ chính xác một coppy mỗi node trong graph.

Kế tiếp, chúng ta hãy đếm số lượng Wine yeey thích từ mỗi người. Chúng ta có thể lấy tất cả các đỉnh và đếm mỗi tên.

Gremlin> wines\_count = [:]

Gremlin> g.V.outE(‘likes’).outV.name.groupCount(wines\_count)

Gremlin>wines\_count

==> Alice = 1

==> Tom = 2

Và như chúng ta mong đợi, Alice thích 1 wine và Tom thích 2 wine.

#### 2.7 Getting Groovy

Bên cạnh các bước Gremlin, chúng ta cũng lấy mảng rộng của ngôn ngữ Groovy các construct và method. Groovy có function map với tên collect() và giảm thiểu function tên inject(). Sử dụng chúng, chúng ta thực hiện mapreduce như các truy vấn.

Cân nhắc trường hợp chúng ta muốn đếm nhiều loại rượu không được xếp hạng. Chúng ta có thể làm từ việc mapping đầu tiên bên ngoài danh sách giá trị cho biết true/false cho dù mỗi rượu có đánh giá. Sau đó, chúng ta chạy danh sách thông qua giảm đếp true và false. Một phần mapping sử dụng collect:

Rated\_list = g.V.in(‘grape\_type’).collect{

!it.inE(‘reported\_on’).toList().isEmpty()

}

Trong đoạn code trên, g.V.in(‘grape\_type’) trả về tất cả các node đến có relationship là grape\_type. Chỉ mỗi rượu có loại của cạnh, vì vậy chúng ta có tất cả rượu trong hệ thống. Kế đến sử dụng collect, chúng ta xác định có cạnh reported\_on. Method toList() gọi tập trung các danh sách đúng, điều này có thể kiểm tra rỗng. rated\_list cung cấp danh sách giá trị đúng, sai.

Để đếm có bao nhiêu rượu không được đánh giá, bạn có thể sử dụng inject()

Rated\_list.inject(0){

Coutn, is\_rated->

If(is\_rated){

count

}else{

Count + 1

}

}

==> 2

Với tất cả các công cụ hiện tại, bạn có thể lấy nhiều sức mạnh biên dịch đồ thị và chuyển đổi. Giả sử bạn muốn tìm cặp bạn trong graph. Để làm điều này, đầu tiên bạn cần tìm tất cả các cạnh với loại friends và sau đó tên output của người cùng nhau chia sẻ cạnh sử dụng transfrom.

g.v.outE(‘friends’).transfrom{[it.outv.name.next(), it.inV.name.next()]}

==>[Patty, Tom]

==>[Patty, Alice]

Để tìm tất cả mọi người và rượu họ thích, chúng ta transform đầu ra cảu mọi người bên trong danh sách với hai thuộc tính: name của mỗi người và danh sách rượi yêu thích.

g.V.both(‘friends’).deup.transfrom{

[it.name, it.out(‘likes’).name.toList()]

}

==>[Alice, [Prancing Wolf Ice Wine 2007]

==> [Patty, []]

==> [Tom, [Prancing Wolf Ice Wine 2007, Prancing Wolf Kabinett 2002]]

Gremlin chắc chắc phải mất một ít lâu để sử dụng, đặc biệt nếu bạn không hoàn thành ngôn ngữ lập trình Groovy trước. Một khi bạn nhận được bên trong sâu vào, bạn sẽ tìm thấy cách thể hiện và các cách ạnh mẽ để truy vấn trong Neo4J.

#### 2.8 Domain-specific steps

Graph traversal thật tuyệt, nhưng các doanh nghiệp và tổ chức có xu hướng trò chuyện trong ngôn ngữ domain-specific. Ví dụ, bạn không muốn hỏi thông thường như Đỉnh nào với cạnh đến là grape\_type chia sẻ cạnh đầu ra của đỉnh wine? Nhưng có thể “Loại rượi nào giống rượi vang?”

Gremlin là ngôn ngữ cụ thể để miền truy vấn dữ liệu đồ thị, nhưng về việc ngôn ngữ có thể làm cụ thể hơn? Gremlin để chúng ta làm điều này từ tạo các bước mới từ các ngữ nghĩa có ý nghĩa đến dữ liệu lưu trữ trong đồ thị.

Hãy bắt đầu tạo một bước mới với tên là varietal để tìm kiếm câu trả lời cho câu hỏi trước đó. Khi varitetal() gọi đến đỉnh, nó sẽ tìm kiếm cạnh đầu ra với loại grape\_type và bước tới đỉnh liên quan.

Khi chúng ta đi vào Groovy-foo ở đây, chúng ta sẽ đầu tiên tìm kiếm code để tạo bước và miêu tả nó dòng đến dòng



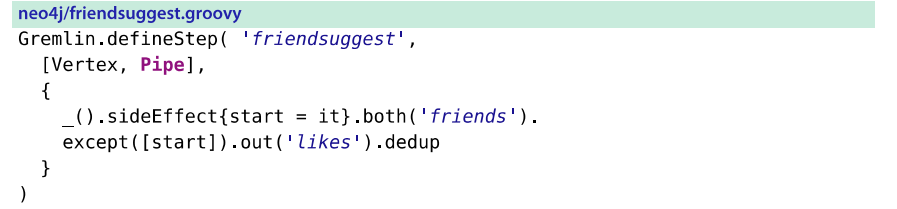
Đầu tiên chúng ta cho biết kỹ thuật Gremlin chúng ta thêm bước mới gọi là varietal. Dòng 2 cho Gremlin biết bước mới này phải đính kèm với Vertex và Pipe class. Dòng cuối là nơi điều kỳ diệu xảy ra. Hiệu quả tạo và mở {} chứa code thực thi. Bên dưới dấu gạch dưới và dấu ngoặc đại diện cho đối tượng pipeline. Từ đối tượng này, chúng ta đến các node bên cạnh liên quan với edge grape\_type, đó chính là varietal node. Cuối cùng dùng dedup để loại bỏ một vài node trùng.

Lời gọi đến bước mới chỉ như các bước khác. Ví dụ, Bên dưới lấy name của ice wine’s varietal:

g.V.filter{it.name == ‘Prancing Wolf Ice Wine 2007’}.varietal.name

==> riesling

Bắt đầu thử một điều khác. Tại thời điểm chúng ta tạo một bước cho yêu cầu hành động thông thường: lấy tất cả rượu bạn yêu thích.



Chỉ sau đó, chúng ta nhận được Gremlin một step mới tên friendsuggest và gắn với Vertex và Pipe. Tại thời điểm, mã của chúng ta sẽ theo dấu người hiện tại. Chúng ta sẽ làm từ việc thiết lập vertex/pipe để các biến sử dụng function sideEfffect{start=it}. Sau khi lấy tất cả các node friends, ngoại trừ người hiện tại.

Bây giờ chúng ta sử dụng với pipe! Chúng ta có thể gọi một step mới như bình thường mong muốn.

g.V.filter{it.name==’Patty’}.friendsuggest.name

==> Prancing Wolf Ice Wine 2007

==> Prancing Wolf Kabinett 2002

Từ varietal và friendsugges chỉ xây dựng Pipe step, bạn cần liên kết chúng cùng nhua để làm cho truy vấn thú vụ hơn. Bên dưới tìm kiếm varietal như bạn của Patty:

g.V.filter{it.name=’Patty’}.friendsugges.varietal.name

==> riesling

Sử dụng metaprogramming Groovy để tạo step mới hiệu quả mạnh mẽ cho tạo domain-specific languages. Nhưng như chính gremlin, việc thực hành có thể mất một vài thứ để sử dụng.

#### 2.9 Update, Delete, Done

Bạn có thể insert và step thông qua đồ thị, nhưng về update và delete dữ liêu. Nó dễ hơn nhiều, một khi bạn tìm thấy vertex hoặc edge bạn muốn sửa đổi. Hãy thêm cân nặng để biết Alice thích Prancing Wolf Ice Wine 2007 đến bao nhiêu.

Gremlin> e=g.V.filter{it.name==’Alice’}.outE(‘likes’).next()

Gremlin> e.weight = 95

Gremlin> e.save

Chúng ta có thể loại bỏ giá trị dễ dàng.

Gremlin>e.removeProperty(‘weight’)

Gremlin>e.save

Đối tượng graph có function để remove verticel và edges, removeVertex và removeEdge, tương ứng. Chúng ta có thể hủy graph từ việc loại bỏ tất cả verticel và edges

Gremlin> g.V.each{g.removeVertex(it)}

Gremlin> g.E.each{g.removeEdge(it)}

Bạn có thể xác nhận chúng có hoạt động bằng việc gọi g.V và g.E. Hoặc bạn có thể hoàn thành điều tương tự với clear() method.

Gremlin> g.clear()

Nếu như bạn chạy các instance của Gremlin (bên ngoài web interface), nó là một ý tưởng tốt để shut down graph connection với shutdown() method.

Gremlin>g.shutdown()

Nếu bạn không làm, nó có thể hỏng dữ liệu. Nhưng thường nó sẽ chỉ cảnh báo bạn lần tới kết nối graph.

#### 2.10 Wrapup

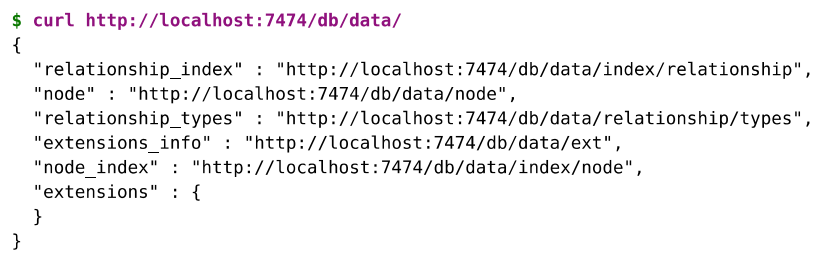
<http://stackoverflow.com/questions/19837065/how-to-install-neo4j-2-0-as-a-windows-service>

<http://romikoderbynew.com/2011/06/11/neo4j-and-gremlin-plugin-install-guide/>

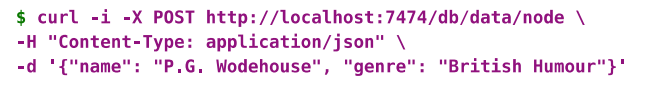
### Tìm hiểu về Rest, chỉ mục, thuật toán(Mục 7.3)

#### 3.1 Rest:

* Tổng quan: Rest là interface có thể sử dụng để tạo node(nút) và relationship(quan hệ giữa các nút). Sau đó dùng rest để đánh chỉ mục và thực hiện tìm kiếm các node mong muốn. Ở đây chúng ta sẽ cùng tìm hiểu cách sử dụng Gremlin query để truy vấn thông qua Rest API.
* Hiểu về rest interface: Bước đầu kiểm tra xem Rest server có đang chạy hay không bằng
* Chạy đoạn mã để kiểm tra server hoạt động hay không, kết quả sẽ trả về dạng json mổ tả url được gọi.



* Tạo node bặng rest:
* Gửi một phương thức post đến /db/data/node path với dữ liệu json:

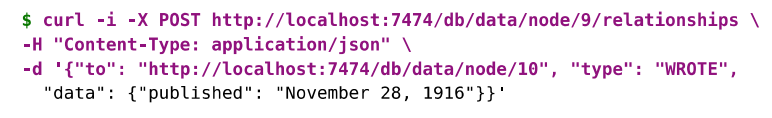


Nếu muốn them thuộc tính cho một node. Có thể dùng GET để them thuộc tính vào URL của node hoặc với nhưng property đặc biệt thì them vào tên thuộc tính.

Ví dụ: Curl <http://localhost:7474/db/data/node/9/properties/genre> “British Humour”.

Tương tự thêm nhiều node với các thuộc tính [“name” : “Jeeves Takes Charge”, “tyle” : “short story”].

Thêm mối qua hệ giữa hai node thể hiện P. G. Wodehouse viết truyện ngắn “Jeeves Takes Charge”.

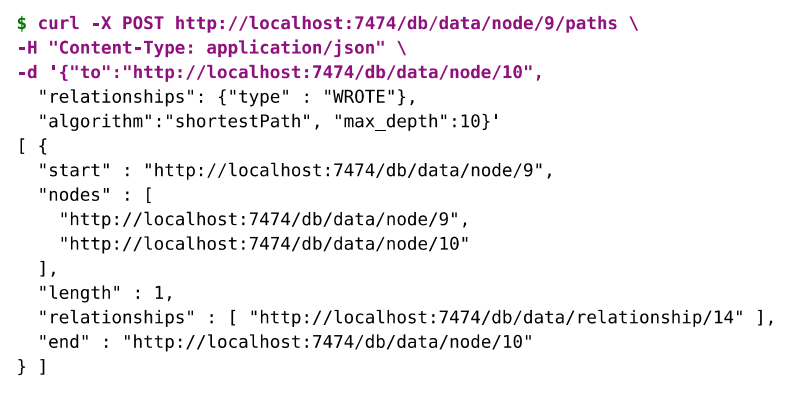


Điểm hay của REST interface là mô tả cách tạo quan hệ sớm trong thân của thuộc tính metadata’s create\_relationship

* Tìm đường (Finding your path):

Thông qua rest, bạn có thể tìm đường giữa hai node bằng cách gửi một request dữ liệu để bắt đầu đường dẫn URL bằng phương thức POST với kiểu JSON thể hiện thông tin node muốn tìm, loại relationship, thuật toán tìm kiếm được sử dụng.

Ví dụ: Chúng ta đang tìm một quan hệ WROTE từ 1 node dùng thuật toán ShortesPath

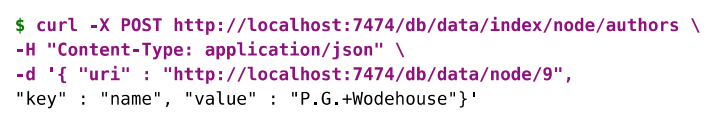


Ngoài ra có thể dùng các thuật toán như allPaths, allSimplePaths, diskstra.

#### Index( Đánh chỉ mục):

Giống như nhiều cơ sở dữ liệu khách, NEO4J đánh chỉ mục để hỗ trợ việc tìm kiếm nhanh hơn. Nhưng không giống những csdl đó, NEO4J đánh chỉ mục phúc tạp hơn vi cần nhiều chỉ mục để thể hiện mối quan hệ với các node khác nhau, chỉ vì vậy mà Indexes Service được tách riêng.

Một cách chỉ mục đơn giản là sử dụng key value hoặc loại hash. Khi sử dụng key value thì key dùng lưu chỉ mục, value lưu REST URL chỉ đến node tương ứng trên đồ thị. Có thể có nhiều indexes vậy nên bạn hãy them tên indexes vào URL:



Để lấy được dữ liệu chỉ đơn giản gọi index là “name” đã đánh ở ví dụ trên:

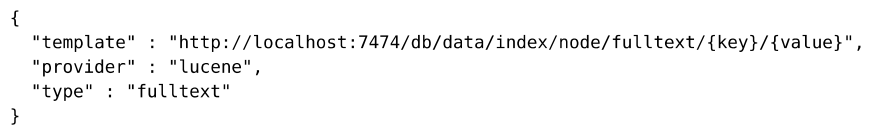


Tương tự như key-value, chúng ta có thể xây dựng index với full-text search. Ví dụ dưới đây đánh chỉ mục để lấy tất cả các cuốn sách có tên bắt đầu với “Jeeves”.

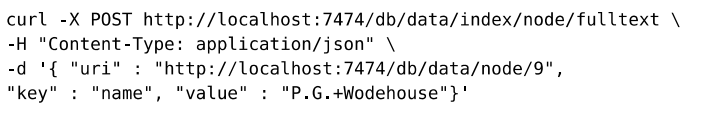
Trước hết xây dựng inverted index:



Sau đó POST sẽ tra về JSON:



Nếu add Wodehouse vào fulltext chung ta được:



Bây giờ việc tìm kiếm Lucene đơn giản gọi:



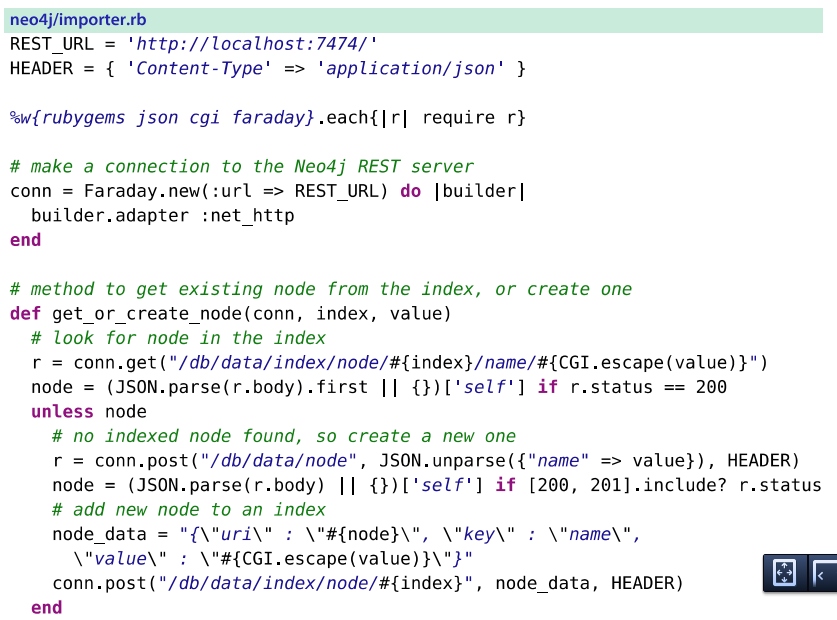
Chỉ mục cũng có thể được xây dựng trên cạnh nối các node, thay các node bang relationship.

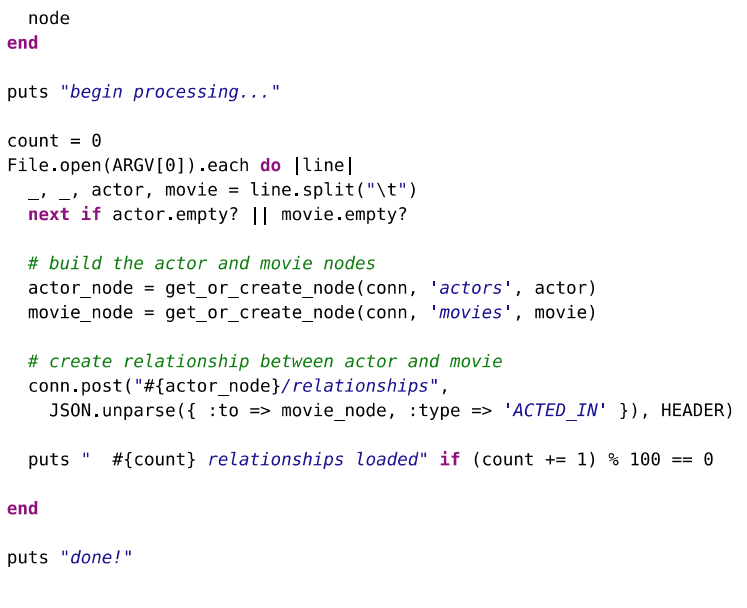
##### 3.2.1 Rest and Gremlin:

Chúng ta đang phân vân là nên dùng Rest interface hay là truy vấn bằng Gremlin, nhưng thực sự thì REST đã cài sẳn plugin Gremlin, cho nên có thể gửi thông qua rest các câu lệnh thực hiện trên Gremlin console. Nó giúp bạn có thể dễ dàng sử dụng, linh động với hai cách với Gremlin sử dụng sức mạnh của truy vấn, REST là khả năng triễn khai và ngôn ngữ linh động

##### 3.2.2 BIG DATA:

Tìm hiểu về hiểu quả khi sử dụng NEO4J với big data. Thử với dữ liệu từ trang web phim Freebase.com. Chúng ta sẽ sữ dụng “perfomance”tab-separated. Dowload script từ đường dẫn về, chúng tạo ra các dòng với các quan hệ giữa các node. Có thể được tìm kiếm bằng chỉ mục name. Dữ liệu có thể tồn tại số lượng lớn thông tin phim, nên phải cài đặt ruby hoặc JSON để chạy script.





Sau khi cài đặt thiết lập xong ruby thì chỉ việc chỉ đến file lưu dữ liệu vừa tải và chạy:

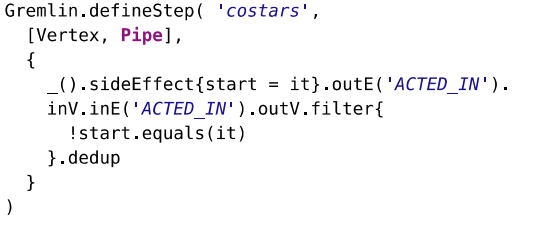


#### 3.3 Algorithms( Thuật toán):

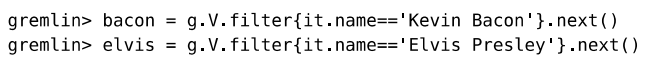
##### 3.3.1 Fancy Algorithms:

Với big data vừa chạy chúng ta sẽ dừng Rest một lúc để quay trở lại với Gremlin.

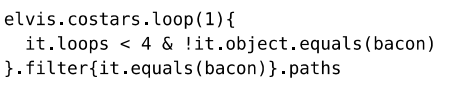
Thuật toán Kenvin Bacon: Là thuật toán đồ thì nội tiếng. Đây là thuật toán tìm khoảng cách ngắn nhất giữa hai actor được sử dụng trong ngành điện ảnh đầu tiên. Chúng ta sẽ xây dựng ví dụ tìm bạn diễn của một diễn viên. Trước hệt chạy Gremlin console và chạy đồ thị sau đó xây dựng costars( bạn diễn) custom theo code dưới đây:



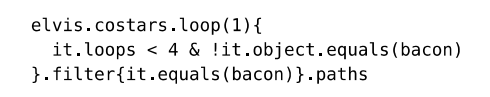
Trong NEO4J chúng ta không dùng nhiều truy vấn cho viết thiết lập dữ liệu. Điểm hay là từ node đầu tiên sẽ chạy qua node gần đó nhất (khoảng cách cạnh và nút). Dưới đây thể hiện điểm bắt đầu và điểm kết thúc.



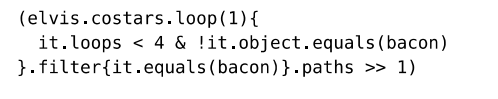
Thêm điều kiện break khỏi vòng lặp:



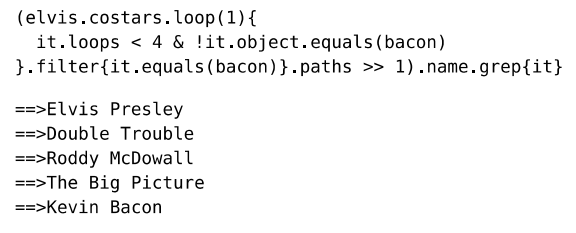
Chỉ để chắc chắn chúng ta không muốn tiếp lục lặp thuật toán Kenvin Bacon cho lần thứ 2, chuyển nút bacon qua vòng lặp ngắn. Nói cách khác là vòng lặp xãy ra nếu chưa đặt 4 lần, chúng ta chưa tìm được bacon node. Sau đó chúng ta xuất đường dẫn để đến được mỗi nut bacon tiếp theo.



Như vậy, chúng ta chỉ đẩy đường dẫn đầu của list nhiều đường dẫn khả dụng, tức là đường ngắn nhất được đến tại vị trí 1. Dùng >> để đẩy item đầu tiên ra khỏi list tất cả các node.



Cuối cùng chúng ta có thể lấy tên của mỗi đĩnh đầu và lọc ra bất ki cạnh không có giá trị bằng Groovy grep command.



Thuật toán Random walk:

Khi tìm kiếm đối tượng tốt từ một tập dữ liệu lớn, cách hữu dụng là “random walk”. Bắt đầu với một số :

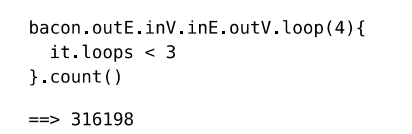
Rand = new Random();

Sau đó lọc ra tỉ lệ đích trên tổng số. Nếu chúng ta muôn lây về chỉ 1/3 của Kenvin Bacon xấp xĩ 60 bộ film, chúng ta có thể lọc bất kì số random nào nhỏ hơn 0.33.

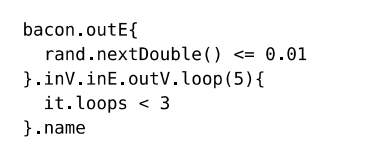
Bacon.outE.filter{rand.nextDouble()<=0.33}.inV.name

Số lượng nên là những nơi bao quanh 20 random.

Gỉa sử đỉnh thứ 2 từ Kevin Bacon. Bạn diễn của bạn diễn anh ấy hoàn toàn là một list(nhiều hơn 300,000 ).



Nếu bạn cần 1 % của list trên, thêm một filter. Chú ý thêm filter là một bước, do vậy bạn sẽ cần thêm một bước nữa vào số vòng lặp.

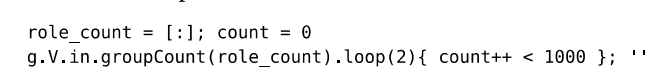


##### 3.3.2 Thuật toán Centrality Park:

Centrality là một các ướng lượng những node độc lập với cả đồ thị. Ví dụ: Nếu chúng ta muốn ướng lượng độ quan trọng của mỗi nút trong mạng la dựa vào khoảng cách của nó so với tất cả các node còn lại. Nó yêu cầu sử dụng thuật toán centrality.

Thuật toán centrality nỗi tiếng nhất chỉ có thể là Google’s PageRand, nhưng có một vài kiểu. Chúng ta sẽ thực thi một version đơn giản gọi là eigenvector centrality, no chỉ đếm số cạnh vào ra liên quan đến một node. Chúng ta sẽ cho mỗi actor 1 con số liên quan đến bao nhiêu vai diễn họ có.

Chúng ta cần một bản đồ cho groupCount() để thực hiện và một count để đặt giá trị lặp cực đại.



Muốn sắp xếp output dùng lệnh: role\_count.sort{a,b -> a.value ⬄ b.value}

Kết quả cuối cùng sẽ là actor với số lược vai diễn lớn nhất.

##### 3.3.3 Thuật toán External Algorithms:

Tự viết thuật toán thì ổn, nhưng đa số những việc đã có sẵn cách làm. Java Universal Network/Graph (JUNG) Framework la một tập hợp các thuật toán đồ thì phổ biến. Nhờ có dự án Gremlin/Blueprint, chúng ta dễ dàng truy cập được nhưng thuật toán của JUNG. Ví dụ: PageRank, HITS, Voltage, centrality, graph-as-a-matrix.

Để dùng JUNG chúng ta cần gói Neo4j vào trong một đồ thị JUNG mới. Để truy cập đồ thị JUNG graph, chúng ta cần thực hiện một trong hai cách: Tải và cài đặt Blueprint và JUNG (file jar) vào thư mục Neo4j server libs sau đó khởi động lại server, hoặc tải gói cài đặt trước Gremlin console.

Gỉa sử bạn đã tải xong gremlin console, đóng neo4j server và khởi đọng Gremlin. Bạn sẽ phải tạo đối tượng Neo4jGraph và chỉ nó đến thư mục cái đặt data/graph.

G = new Neo4jGraph(‘/users/x/neo4j-enterprise-1.7/data/graph.db’)

Chúng ta sẽ giữ Gremlin graph tên là g. Neo4jGraph object cần được gói lại trong GraphJung object:

J= new GraphJung( g )

Một phần lí do Kenvin Bacon được chọn giống nhưu đường dẫn đích cơ bản là vì mối qua hệ gần với các actor khác. Một điều qua trọng, anh ấy không cần nhiều vai diễn nhưng đơn giản là có liên kết với những người nhiều vai diễn.

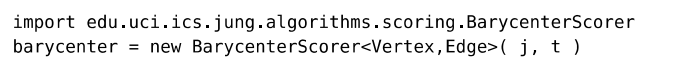
Có một câu hỏi phát sinh: chúng ta có thể tìm một actor tốt hơn Kenvin Bacon?

JUNG chưa thuật toán tính điểm gọi là BarycenterScorer cái cho một số điểm đến mỗi đĩnh dựa vào khoảng cách đến các đỉnh khác. Nếu Kenvin Bacon là lựa chọn tốt nhất, chúng ta sẽ mông đợi điểm của anh ấy sẽ là nhỏ nhất, nghĩa là anh ấy gần nhất so với các actor còn lại.

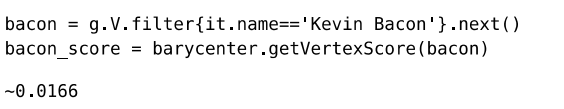
Thuật toán JUNG của chúng ta nên áp dụng cho tập nhiều actor, do vậy chúng ta có thể cài một transformer để tìm chỉ một node. EdgeLabelTransformer cho phép chỉ những node đó với một cạnh của ACTED\_IN đến thuật toán:

T = new EdgeLableTransformer([ACTED\_IN’)] as Set, false)

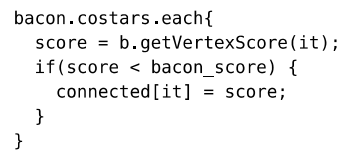
Tiếp theo chúng ta cần import thuật toán, truyền vào GraphJung and Transformer.



Với nó, chúng ta có thể lấy điểm BarycenterScorer của nhiều node. Hãy tìm ra những điểm đó:



Khi chúng ta có điểm của Kenvi Bacon chúng ta có thể duyệt qua mỗi đỉnh và lưu bất kì đỉnh nào có điểm thấp hơn. Nó có thể mất một thời gian dài để chạy điểm BarycenterScorer cho mỗi actor trong database. Do vậy, chúng ta chỉ chạy thuật toán với mỗi bạn diễn của Kenvin. Nó có thể mất vài phút, tùy thuộc vào phần cứng. BarycenterScorer thí nhanh, nhưng chậm ở quá trình chuyễn đổi giữa các bạn diện được thêm vào:



Tất cả các key tồn tại trong bản đồ kết nối biểu hiện một lựa chọn tốt hơn Kenvin Bacon. Nhưng nó tốt để có một cái tên chúng ta dễ đọc, do vậy output tất cả các node và lấy một cái chúng ta thích.

Với j graph vừa tạo ở trên, chúng ta có thể chạy bất kì thuật toán JUNG nào trong dataset, nhưn: PageRank.

Dang sách tất cả các thuật toán JUNG có thể tìm trên Javadoc API online.

### Distributed High Availibality (Mục 7.4 tính sẵn có cao được phân bổ )

#### 4.1 Transaction (giao tác)

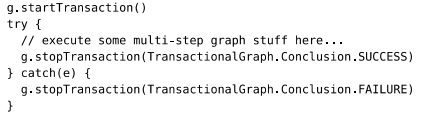
Neo4j là một cơ sơ dữ liệu giao tác ổn định, độc nhất, chắc, nguyên tử , giống PostgreSQL. Diều này làm cho nó lựa chọn tốt cho dữ liệu quan trọng, bạn có thể chọn một cơ sở dữ liệu quan hệ khác. Chí giống như các giao tác bạn đã thấy trước đó, giao tác Neo4j là những hoạt hoạt động tất cả hoặc không có gì. Khi một giao tác bắt đầu, mỗi hoạt động sau đây sẽ thành công hoặc thất bại như là một đơn vị nguyên tử- thất bại của một có nghĩa là sụ thất bại của tất cả.

Chi tiết về cách thức các giao tác được xử lý vượt xa Gremlin vào dự án Neo4j cơ bản gọi là Blueprint. Chúng ta dùng gremlin 1.3 cái mà nó sử dung Blueprint 1.0 . Nếu bạn sử dụng một phiên bản khác, ban có thể tìm kiếm chi tiết ở Blueprint API Javadocs.

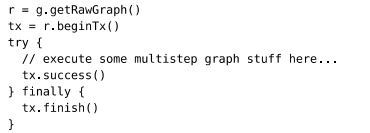
Giống như PostgreSQl, các dòng hàm cơ bản là tự động bao phủ một giao tác ẩn. Để giải thích giao tác đa dòng, chúng ta cần dố tượng cờ để tắt tự động phương thức giao tác, theo Neo4j chúng ta lên kế hoạch xử lý giao tác bằng tay. Ban có thể đổi phu87o7ng thức giao tác sang hàm *setTransactionMode()*



Bạn bắt đầu và kết thúc giao tác trên đối tượng đố thị sử dụng *startTransaction()* và *stopTransaction(conclusion)*. Khi bạn dừng giao tác, bạn cũng cần đánh dấu cho dù giao tác đó thành công. Nếu không, Neo4j có thể rollback tất cả lệnh thực hiện từ lúc bắt đầu. Nó là một ý tưởng tốt để bao phủ giao tác trong một *try/catch* để dảm bảo chác chắc rằng bất kể ngoại lệ nào trigger rollback



Nếu bạn muốn vận hành bên ngoài phạm vi Gerimlin và làm việc trực tiếp với Neo4j *EmbeddedGraphDatabase,* bạn có thể sử dụng cú pháp Java API cho giao tác. Bạn có thể dủng cách này nếu bạn viết code Java hoặc sử dụng một ngôn ngữ như JRuby



Sự đa dạng của cả hai cung cấp cho bạn đầy đủ những dảm bảo giao tác ACID. Ngay cả khi sự thất bại hệ thống sẽ đảm bảo bất kì mô tả roll back khi máy chủ được kích hoạt trở lại. Nếu ban không muốn diều khiển giao tác bằng tay, bạn tốt nhất nên tắt việc duy tri phương thức giao tác trên *TransactionalGraph.Mode.AUTOMATIC*

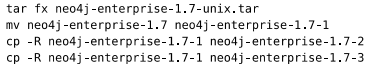
#### 4.2 High Availability

Phương thức có sẵn cao là câu hỏi cần được trả lời của Neo4j, “có thể là phạm vi cơ sở dữ liệu đồ thị ?” có, nhưng với một vài cảnh báo.

#### 4.3 HA Cluster

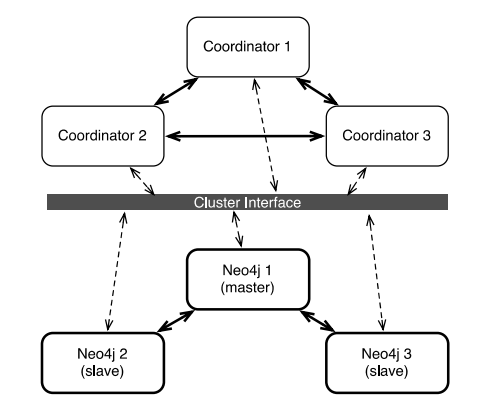
Để sử dụng Neo4j HA, chúng ta phải cài đặt một cluster. Neo4j dùng dich vụ điều phối cluster bên ngoài gọi là Zookeeper. Zookeeper là một đề án xuất sắc khác phát sinh từ dự án Apache Hadoop. Nó là một dịch vụ có mục đích chung để điều phối ứng dung. Neo4j HA sử dụng cái này để quản lý vòng đời hoạt động của nó. Mỗi máy chủ Neo4j có diều phối viên liên quan.

May mắn thay, tập đoàn Ne4j gom chung với Zookeeper như các files để giúp chúng ta định hình một cluster. Chúng ta sẽ chạy 3 trường hợp của Neo4j phiên bản 1.7. ban có thể tải về từ website hệ thống và sau đó giải nén.



Chúng sẽ theo 5 bước để tạo cluster

1. Đặt ID duy nhất cho mổi điều phối viên máy chủ
2. Dịnh hình cho mỗi diều phối viên máy chủ để giao tiếp với các máy chủ khác và host máy chủ Neo4j
3. Khởi động 3 điều phối viên máy chủ
4. Định hình mỗi máy chủ Neo4j để chạy phương thức HA, cho chúng ports duy nhất, và sắp xếp chúng theo nhận biết của cluster



1. Khởi động tất cả 3 máy chủ Neo4j

Zookepper tìm mỗi máy chủ bằng con đường ID duy nhất cho cluster. Số này chỉ là một giá trị trong file *data/coordinator/myid*. Máy chủ 1 chúng ta sẽ giữ mặt định là 1, máy chủ 2 là 2 và đặt máy chủ 3 là 3.



Chúng ta phải chỉ ra một số thông báo cài đặt bên trong cho cluster. Mỗi máy chủ sẽ có một file tên *conf/coord.cfg.* Mặc định, thông báo máy chủ 1 có thể thay đổi máy chủ nhu localhost và 2 port đặt: port chọn theo quy định (2888) và port chọn điều khiển (3888)

#### 4.4 Building the cluster ( xây dựng cluster)

Zookeeper đại diện cần thiết là một nhóm các máy chủ trong cluster và port chúng trao đổi qua nhau. Port master thì được sử dụng khi cái master đi xuống – port đặc biệt được sử dụng cho nên máy chủ có thể chọn một master mới.. Giữ server.1 và thêm server.2 và server.3 để dùng successive port.

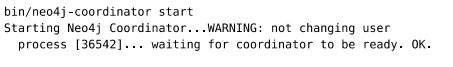


clientPort : 2181 cho server.1

clientPort: 2182 cho sewrver.2

clientPort: 2183 cho swerver.3

Coordinate



#### 4.5 Writing in Neo4j ( viết trong Neo4j)

Mở conf/neo4j-server.properties và thêm vào dòng dưới mỗi máy chủ:

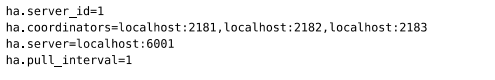


Thông thường port mặc định 7474 là ổn, nhưng khi chúng ta chạy 3 trường hợp Neo4j trên 1 hộp, chúng ta không thể khiến chúng lấn http/https. Chọn port 7471/7481 cho server.1, 7472/7482 cho server.2, 7473/7483 cho server.3

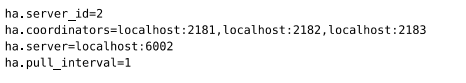


Port Neo4j server sẽ trao đổi thông tin vói nhau

Server 1 thêm vào dòng:



Server 2:



Verifying cluster status (kiểm tra trang thái cluster)

Có thể kiểm tra bằng cách mở gán Neo4j trường hợp web admin (server 1 port 7471). Chọn link Server Info ở trên và High Availability trên menu

Thộc tính dưới High Availability liệt kê thông tin về cluster này. Nếu server là master server thì thuộc tính là true. Nếu không, có thể tiìm server chon master dưới InstanceInCluster.

Verifying Replication (sao chép kiểm tra)

Vói cluster đang chay, có thể kiểm tra server của ban đang sao chép đúng. Nếu tất cả đều đi đúng kế hoạch, 1 slave truyền đến node master và cuối cùng là đến các server slave khác. Nếu mở web console cho mỗi server, có thể sử dụng gắn liền Gremlin console ở web admin. Đối tương đồ thị Gremlin thay đổi để phủ HighlyAvailableGraphDatabase



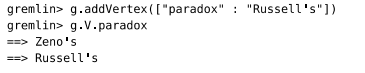
Kiểm tra server



Chọn server master console và đầu ra giá trị nghịch lý cao nhất



Nếu chọn server slave và thêm nghich lý Russell



#### 4.6 Master election ( chọn master)

Nếu tắt server master và nạp lại server cũ, thì sẽ thấy server khác dã được chọn làm master mới. khởi động lại server thêm nó về lại cluster, nhưng bây giờ master cũ vẫn sẽ là một slave

Tính có sẵn cao cho phép đọc năng hệ thống để phân chia sao chép đồ thị qua các server phức tạp và chia sẻ gánh nặng.

#### 4.7 Backups (hỗ trợ)

Tập đoàn Neo4j đưa ra một công cụ hỗ trợ đơn giản là neo4j-backup

Phương pháp mạnh mẽ nhất khi chạy server HA có câu lệnh hỗ trợ đầy đủ để sao chép cơ sơ dữ liệu từ cluster đến file đóng dấu ở ổ mount. Mỗi server trong cluster sẽ đảm bảo bạn lấy đuộc dữ liệu có sẵn gần nhất

Phải khởi động tất cả hỗ trợ. Đây là hỗ trợ cluster HA cho 1 danh bạ



Nếu không chạy trên phương thức HA, chỉ đổi phương thức trong URI thành single



### 5. Tổng kết (Mục 7.5)

Neo4j là thực hiện mã nguồn mở của các lớp của đồ thị cơ sở dữ liệu. Đồ thị cơ sở dữ liệu tập trung vào các mối quan hệ giữa các dữ liệu, chứ không phải là tính chung của các giá trị. Mô hình hóa dữ liệu đồ thị là đơn giản. Bạn chỉ cần tạo các nút và các mối quan hệ giữa chúng và tùy chọn treo cặp giá trị từ chúng. Truy vấn là dễ dàng giống như khai báo cách như thế nào để đi trên đồ thị từ một nút bắt đầu.

Thế mạnh của Neo4j

Neo4j là một trong những ví dụ tốt nhất của mã nguồn mở cơ sở dữ liệu đồ thị. Cơ sở dữ liệu đồ thị là hoàn hảo cho dữ liệu phi cấu trúc, bằng nhiều cách thậm chí hơn cả kho dữ liệu tài liệu. Không chỉ là kiểu và lược đồ Neo4j, nhưng nó không bắt phải ép buộc về cách dữ liệu có liên quan. Đó là trong ý nghĩa tốt nhất, miễn phí cho tất cả. Hiện nay, Neo4j có thể hỗ trợ 34,4 tỷ nút và 34.4 tỷ mối quan hệ, đó là quá đủ cho hầu hết sử dụng.

Sự phân bố Neo4j cung cấp một số công cụ để tra cứu nhanh chóng với Lucene và mở rộng ngôn ngữ dễ sử dụng như Gremlin và giao diện Rest. Beyond dễ sử dụng, Neo4j là thực tế, không giống như tham gia các hoạt động trong cơ sở dữ liệu quan hệ hoặc bản đồ giảm hoạt động trong cơ sở dữ liệu khác, đi ngang qua đồ thị là thời gian liên tục. Dữ liệu chỉ là một nút bước đi, chứ không phải là giá trị tham gia với số lượng lớn và lọc các kết quả mong muốn - như hầu hết các cơ sở dữ liệu, chúng ta đã nhìn thấy hoạt động. Nó không quan trọng cách các đồ thị lớn sẽ trở thành: di chuyển từ nút A đến nút B luôn luôn là một bước là nếu chúng chia sẻ mối quan hệ. Cuối cùng, phiên bản Enterpirse cung cấp cho tính sẵn sàng cao.

Điểm yếu của Neo4j

Neo4j có một vài thiếu sót. Các cạnh trong Neo4j không thể chỉ dẫn một đỉnh trở lại trên chính nó. Chúng tôi cũng tìm thấy sự lựa chọn của danh pháp để thêm phức tạp khi giao tiếp. Mặc dù HA là tuyệt vời với nhân bản, nó chỉ có thể tái tạo đầy đủ đồ thị đến các máy chủ khác. Nó hiện không thể Shard đồ thị con, mà vẫn đặt một giới hạn về kích thước đồ thị. Cuối cùng, nếu bạn đang tìm kiếm một giấy phép mã nguồn mở dành cho doanh nhân, Neo4j có thể không được cho bạn. Trường hợp các phiên bản của cộng đồng là GPL, nếu bạn muốn chạy một môi trường sử dụng các công cụ doanh nghiệp, có thể bạn sẽ cần phải mua một giấy phép.

Neo4j CAP

Nếu bạn chọn để phân phối, tên gom cụm "sẵn sàng cao" nên cho đi chiến lược của họ. Neo4j HA có sẵn và phân vùng chịu. Mỗi nút phụ thuộc sẽ chỉ trả về những gì hiện có , có thể đồng bộ với các nút chính tạm thời. Mặc dù bạn có thể làm giảm độ trễ cập nhật bằng cách tăng khoảng kéo một nút phụ thuộc, nó vẫn còn kỹ thuật cuối cùng phù hợp. Đây là lý do tại sao Neo4j HA được khuyến khích cho hầu hết các yêu cầu đọc

Chia sẻ suy nghĩ

Đơn giản Neo4j có thể được off-putting nếu bạn đang không được sử dụng để mô hình hóa dữ liệu đồ thị. Nó cung cấp một nguồn API mở mạnh mẽ với nhiều năm sử dụng sản xuất nhưng vẫn có khá nhiều người sử dụng. Chúng tôi viết bằng phấn này lên đến thiếu hiểu biết, kể từ khi cơ sở dữ liệu đồ thị lưới để tự nhiên cách con người từ mười đến khái niệm hóa dữ liệu. Chúng ta tưởng tượng gia đình của chúng ta là cây, hoặc bạn bè của chúng ta như đồ thị: hầu hết chúng ta không tưởng tượng mối quan hệ cá nhân như kiểu dữ liệu tự tham chiếu. Đối với các lớp nhất định của các vấn đề, như các mạng xã hội. Neo4j là một sự lựa chọn rõ ràng. Nhưng bạn nên cung cấp cho nó một số xem xét nghiêm túc cho vấn đề hiển nhiên là tốt - nó chỉ có thể làm bạn ngạc nhiên như cách mạnh mẽ và dễ dàng .