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): Giới thiệu tổng quan về việc lưu trữ dữ liệu bằng đồ thị, các loại Graph store và đi cụ thể vào một loại Graph store là Neo4j.

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 mở đầu, 7.1 và 7.5 topic 3b | 1212205 – Nguyễn Phượng Lĩnh | 100% | 9/10 |
| Tìm hiểu mục 7.1, 7.2 topic 3b | 1212209 – Nguyễn Đức Hoàng Long | 100% | 9/10 |
| 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 2](#_Toc437198714)

[Nội Dung: 3](#_Toc437198715)

[I. Graph store 3](#_Toc437198716)

[1. Tổng quan 3](#_Toc437198717)

[1.1. Khái niệm 3](#_Toc437198718)

[1.2. Đặc điểm 3](#_Toc437198719)

[2. Liên kết dữ liệu ngoài với chuẩn định dạng mô tả tài nguyên (Resource Description Format- RDF) 4](#_Toc437198720)

[3. Tình huống sử dụng cơ sở dữ liệu dạng đồ thị (Use case for graph stores) 4](#_Toc437198721)

[II. NEO4J 9](#_Toc437198722)

[1. Neo4j là bảng thân thiện (Mục 7.1) 10](#_Toc437198723)

[2. Graph, Groovy, Cypher và CRUD (Mục 7.2) 12](#_Toc437198724)

[3 Tìm hiểu về Rest, chỉ mục, thuật toán (Mục 7.3) 29](#_Toc437198725)

[4. Distributed High Availibality (Mục 7.4 tính sẵn có cao được phân bổ ) 38](#_Toc437198726)

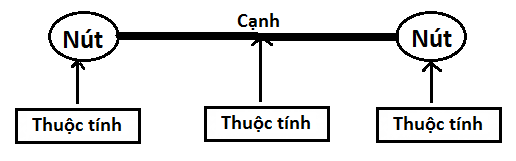
[5. Tổng kết (Mục 7.5) 43](#_Toc437198727)

[III. Tài liệu tham khảo 44](#_Toc437198728)

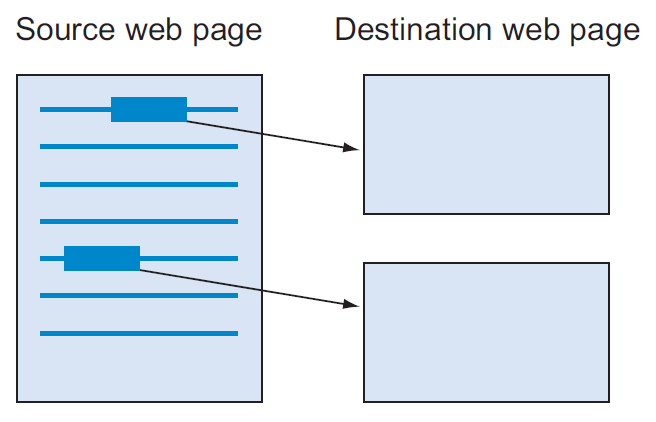
# Nội Dung:

## Cơ sở dữ liệu dạng đồ thị (Graph stores)

### Tổng quan

* 1. Khái niệm
* Cơ sở dữ liệu dạng đồ thị là một cơ sở dữ liệu sử dụng cấu trúc đồ thị với các nút, các cạnh, và các thuộc tính để mô tả và lưu trữ những dữ liệu có mối quan hệ chặt chẽ với nhau. Trong đó:
  + Các nút (node) đại diện cho các đối tượng trong thế giới thực là các danh từ. Ví dụ tên của người, các tổ chức, các trang web, … hoặc thậm chí là các tế bào sinh học trong cơ thể.
  + Các cạnh (edge) là các đường kết nối giữa các nút và là đại diện cho mối quan hệ (Relationship) giữa các đối tượng.
* Thuộc tính (property) là các định danh của các nút và các cạnh, giúp làm tăng tính ngữ nghĩa cho các cạnh, các nút .
* 
* Hình Cấu trúc của cơ sở dữ liệu dạng đồ thị
  1. *Đặc điểm*
* Cơ sở dữ liệu dạng đồ thị thường được sử dụng khi cần lưu các dữ liệu phức tạp mà lại liên quan mật thiết với nhau.
* Nhờ cấu trúc đồ thị mà khi xác định được 1 nút, thì có thể nhanh chóng tìm ra các nút lân cận của nút đó mà chỉ cần thực hiện câu truy vấn vô cùng đơn giản. Nhờ vậy mà tốc độ tính toán khá là nhanh.
* Có 1 hạn chế của cơ sở dữ liệu dạng đồ thị này là nó khó phân tán cơ sở dữ liệu dạng đồ thị thành nhiều mảnh. Có rất nhiều nghiên cứu trong lĩnh vực này nhưng chưa có bất kì giải pháp nào tin cậy được đưa ra..
* Cơ sở dữ liệu dạng đồ thị thường giải quyết các vấn đề về mạng. Ví dụ như mạng xã hội, …
* Khi truy vấn đến cơ sở dữ liệu dạng đồ thị, kết quả trả ra là 1 đồ thị con gồm các nút, các cạnh, các thuộc tính mà có liên quan đến nội dung truy vấn.

### Liên kết dữ liệu ngoài với chuẩn định dạng mô tả tài nguyên (Resource Description Format- RDF)



Hình Ví dụ minh hoạ cơ sở dữ liệu dạng đồ thị   
mô hình việc liên kết giữa các trang web với nhau

* Trong nhiều trang web hiện nay, thông thường tại 1 trang web thì sẽ có một số đường dẫn dẫn sang trang web khác. Và để mô hình lại sự liên kết giữa các trang web đó, thì sẽ sử dụng cơ sở dữ liệu dạng đồ thị. Tuy nhiên, việc mô hình lại sự liên kết giữa các trang web cũng cần tuân thủ 1 số quy định, những quy định này được gói gọn trong 1 chuẩn gọi là Resource Description Format (RDF).
* Với chuẩn RDF, mỗi URL là định danh cho mỗi nút trong đồ thị, sự liên kết giữa các URL là các cạnh nối các nút lại với nhau.

Các URL liên kết với nhau khá phức tạp, thậm chí 1 số URL còn liên kết với những URL khác ngoài hệ thống.

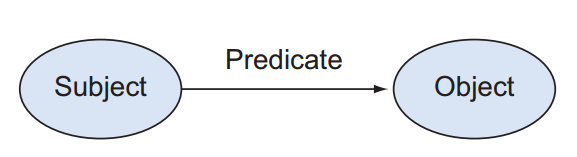
### Tình huống sử dụng cơ sở dữ liệu dạng đồ thị (Use case for graph stores)

#### 3.1 Phân tích liên kết (Link analysis)

* Dựa vào cơ sở dữ liệu dạng đồ thị, chúng ta có thể tìm đến các nút lân cận một nút cho trước một cách nhanh nhất, và có thể xác định các mối quan hệ khác của các nút lân cận đó nhờ vào sự liên kết giữa các nút với nhau.
* Chức năng này của cơ sở dữ liệu dạng đồ thị thường được thấy trong các mạng xã hội với nhau.
* Ngoài ra, nó còn hữu ích trong việc xử lý ngôn ngữ tự nhiên. Ví dụ, với mỗi một câu được nói, thì sẽ có thể có rất nhiều ngữ nghĩa, thậm chí chứa các từ đồng âm và nó phụ thuộc rất nhiều vào các ngữ cảnh, tình huống sử dụng khác nhau. Và để hiểu được, xử lý được, ta cần phải có 1 mô hình thể hiện các ngữ cảnh khác nhau với các ý nghĩa khác nhau của mỗi từ trong 1 câu ấy.
* Bên cạnh đó, cơ sở dữ liệu dạng đồ thị còn được sử dụng để liên kết các dữ liệu lại với nhau và tìm kiếm các mô hình dữ liệu trong các tài liệu lớn. Các thực thể trong các tài liệu ấy là các nút. Việc xác định các thực thể ấy là rất quan trọng. Và sau khi xác định các thực thể ấy, thì dùng cơ sở dữ liệu dạng đồ thị để xây dựng mô hình cho các thực thể ấy, cũng như sử dụng các thực thể đấy để thực hiện tìm kiếm nâng cao.

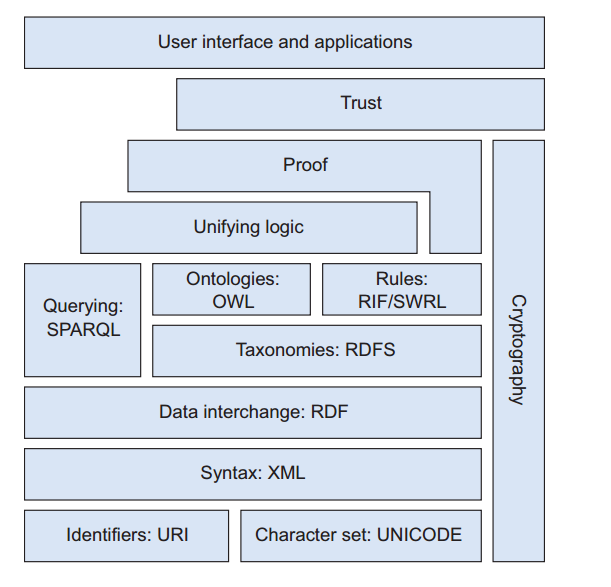
#### 3.2 Sử dụng luật và các suy luận trong đồ thị (Graph, rule, and inference)

* Một cơ sở dữ liệu dạng đồ thị thường sử dụng 1 số thuật ngữ. Các thuật ngữ này thường xuất phát từ các hệ thống logic và ngôn ngữ sử dụng. Những thuật ngữ này được sử dụng để xác định các luật trừu tượng, liên quan đến sự hiểu biết về các đối tượng trong hệ thống, và được thể hiện ở việc mô tả các nút trong đồ thị theo chuẩn RDF. Với các bối cảnh, tình huống, địa điểm sử dụng khác nhau thì thuật ngữ sử dụng cũng khác nhau. Ví dụ 1 số thuật ngữ như: Source, link, destination hoặc Subject, predicate, object.



Hình Minh hoạ sử dụng thuật ngữ cơ sở dữ liệu dạng đồ thị

* RDF là 1 cấu trúc với các mục đích chung, có thể được sử dụg để lưu nhiều hình thức logic. RDF thường được sử dụng để lưu trữ các logic và luật. Và khi đã thiết lập các luật cũng như logic, ta có thể suy luận ra những điều khác trong hệ thống.
* RDF cũng là 1 phần trong kiến trúc mạng ngữ nghĩa (Semantic Web Stack).
* Mạng ngữ nghĩa là một phương pháp cho phép định nghĩa và liên kết dữ liệu một cách có ngữ nghĩa hơn nhằm giúp máy tính và người dùng cộng tác với nhau tốt hơn. Ngoài ra, mạng ngữ nghĩa còn cung cấp một môi trường chia sẻ và tự động xử lý dữ liệu trên máy tính
* Kiến trúc mạng ngữ nghĩa là một tập hợp(stack) các ngôn ngữ. Tất cả các lớp của mạng ngữ nghĩa được sử dụng để đảm bảo độ an toàn và giá trị thông tin trở nên tốt nhất.



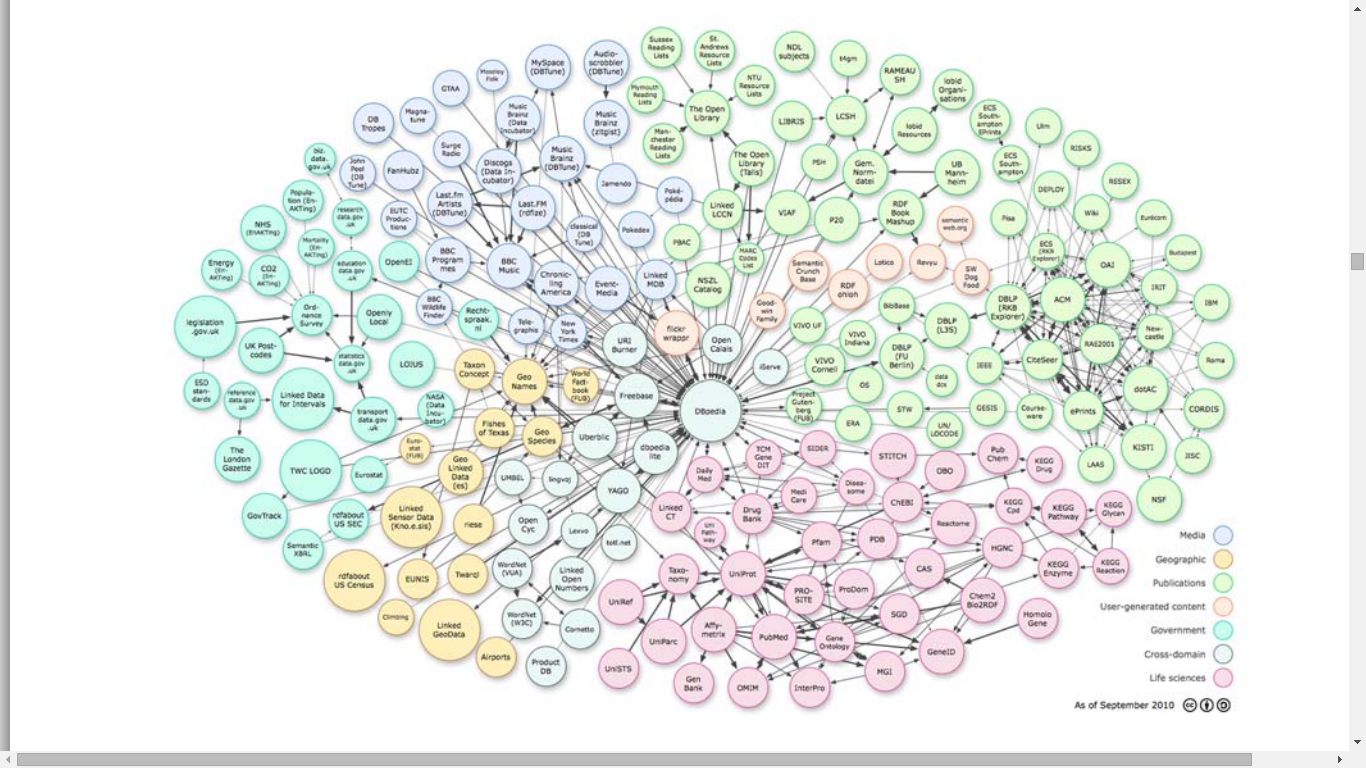
Hình Kiến trúc mạng ngữ nghĩa

.

* Kiến trúc mạng ngữ nghĩa bao gồm các chuẩn cơ bản như URI, XML, RDF. Ngoài ra còn có các chuẩn truy vấn (SPARQL), chuẩn luật (RIF/ SWRL) và còn có User interface and application, và các chuẩn khác.
* Một số chuẩn thì đã được mã hoá, được sử dụng an toàn trong quá trình trao đổi dữ liệu trên internet. Tuy nhiên, còn 1 số chuẩn thì chưa được mã hoá như Logic, Proof, Trust. Và đây cũng là nơi tập trung nhiều nghiên cứu.

#### 3.3 Xử lý các bộ dữ liệu mở (Using graph to process public dataset)

* Như đề cập ở trên, thì các trang web hiện nay thường có các liên kết để dẫn đến các trang web khác khi cần thiết. Và không dừng lại ở đó, đôi khi ta cũng cần truy vấn dữ liệu ở các hệ thống của người khác. Và ta cần 1 kỹ thuật để làm việc này. Đó là kỹ thuật liên kết dữ liệu mở (Linked Open Data – LOD). Kỹ thuật này tập hợp các tài nguyên liên quan để sử dụng thuật toán tự động sinh liên kết, cho phép sáp nhập các bộ dữ liệu khác nhau để tạo ra 1 bộ dữ liệu mới phù hợp với yêu cầu của ta hơn.
* LOD thường được sử dụng trong việc phân tích các dữ liệu trong quá khứ và hiện tại để dự đoán dữ liệu trong tương lại. Từ đó đưa ra các chiến lược, chính sách kinh doanh mới cho phù hợp với tình hình kinh tế hơn.
* LOD còn là 1 tập dữ liệu lớn bao gồm tập dữ liệu nhiều lĩnh vực như kiến thức phổ thông, khoa học đời sống, phương tiện truyền thông, … và 1 số lĩnh vực khác.



Hình Sơ đồ liên kết dữ liệu mở trên nền tảng đám mây (LOD cloud diagram)

* Liên kết dữ liệu mở trên nền tảng điện toán đám mây (LOD cloud) là tất cả các dữ liệu có cấu trúc trong các tập dữ liệu mở, được biểu diễn lại và sau đó xuất bản với các nguyên tắc liên kết dữ liệu.

## NEO4J

**Neo4j là gì?**

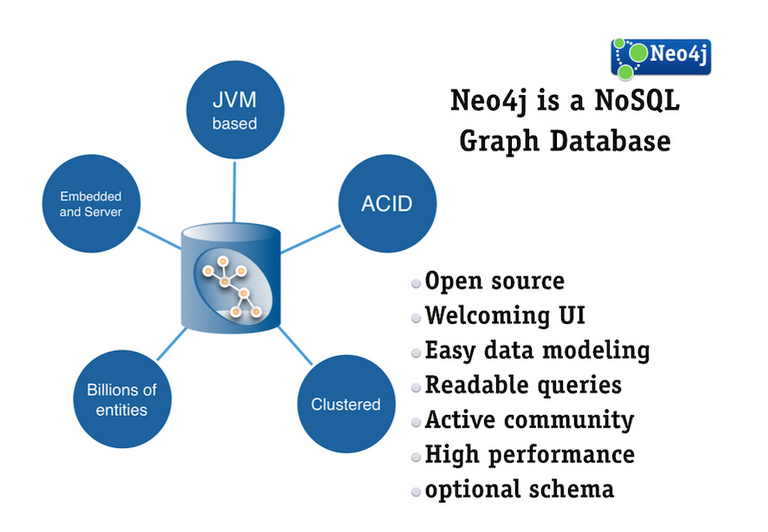
Neo4j là một cơ sở dữ liệu NoSql dạng đồ thị mã nguồn mở xây dựng bằng Java và Scala do tập đoàn Neo technology tài trợ. Được phát triển từ năm 2003 và được sử dụng công khai năm 2007. Neo4j được sử dụng bởi hàng trăm ngàn công ty và tổ chức trong rộng khắp tất cả nghành công nghiệp. Được sử dụng trong quản lý mạng (network management), phân tích phần mềm (software analytics), nghiên cứu khoa học, định tuyến, tổ chức và quản lý dự án, tư vấn, mạng xã hội (social networks)….

**Tính năng nổi bật của Neo4j**

Một số tính năng đặc biệt làm Neo4j trở nên rất phổ biển trong người dùng (user), nhà phát triển (developer) và các DBA (Database Administrator).

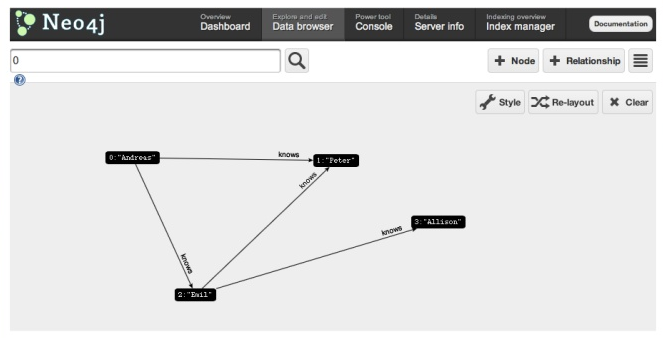
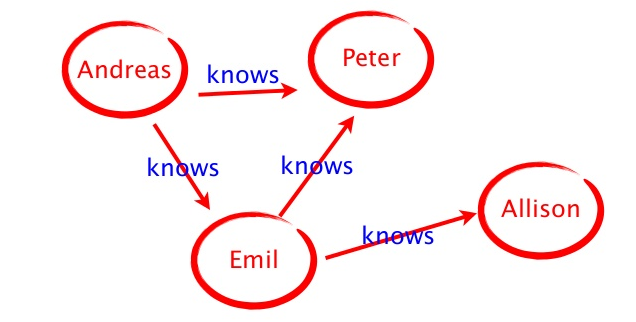
1. Cụ thể hóa các mối quan hệ tại thời điểm tạo, kết quả là không có cho truy vấn thời gian phức tạp.
2. Bỏ qua các hằng số thời gian (constant time traversal) cho các quan hệ kể cả về chiều sâu và chiều rộng do cách biểu diễn các nút và quan hệ có khả năng.
3. Tất cả quan hệ trong Neo4j đều quan trọng và nhanh chóng, khiến nó có thể trở thành thực tế và sử dụng các mối quan hệ mới sau này trên “shortcut” và tăng tốc độ dữ liệu tên miền khi có nhu cầu phát sinh.
4. Lưu trữ nhỏ gọn và bộ nhớ đệm cho các đồ thị, vì vậy có khả năng mở rộng lưu trữ hàng tỉ nút trong một cơ sở dữ liệu trên một hệ thống phần cứng vừa phải.
5. Viết trên các JVM(java virtual machine)

**Sơ đồ tổng quan về Neo4j**



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

**Whiteboard 🡪 Data**



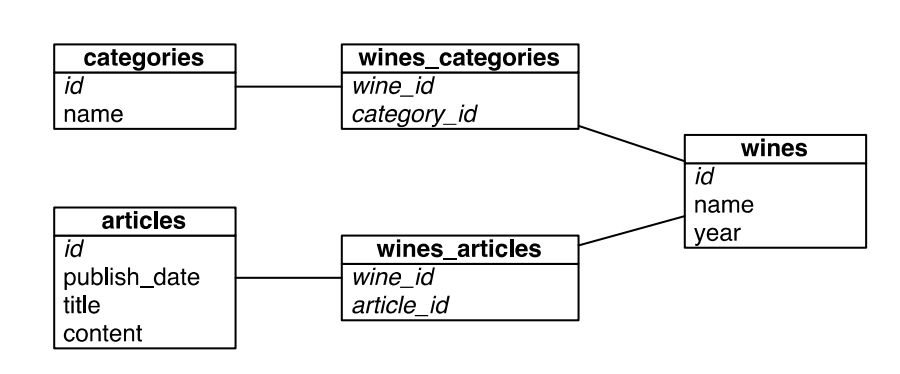
Neo4j được biết đến với “**Whiteboard friendly**”, nghĩa là bạn có thể vẽ các đường và điểm như hình trên, bạn có thể lưu trữ trên Neo4j. Neo4j tập trung nhiều hơn về mối quan hệ giữa các giá trị hơn những điểm giống nhau giữa các thiết lập của giá trị (như tập hợp các tài liệu hay bảng của các dòng). Với cách này việc lưu trữ dữ liệu tự nhiên và dễ hiểu hơn.

**Ví dụ:**

Bạn phải tạo ra được lời đề nghị kỹ thuật rượu với nhiều thành phần khác nhau như: Regions, wineries, vintages và designations.

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…



**Theo lược đồ quan hệ.**



**Theo Neo4j.**

Nhìn vào đồ thị bên trên, ta thấy Neo4J hỗ trợ đầy đủ giá trị và cấu trúc chỉ những thông tin cần thiết để lưu trữ các thông tin trên. Dễ dàng thấy được các thông tin và cấu trúc các quan hệ cần thiết: năm sản xuất 2007, rượu với tên là prancing wolf, loại rượu riesling, thông tin dựa trên wine expert.

**1 số từ đề cập bên trên về rượu:**

* Regions: Vùng miền sản xuất.
* Vineries: Nhà máy sản xuất rượu.
* Vintages: Nơi sản xuất nho.
* Designations: Tên của rượu.
* Riesling: một loại rượu nho nước Đức.
* Wine expert: chuyên gia thẩm định rượu.

### Graph, Groovy, Cypher và CRUD (Mục 7.2)

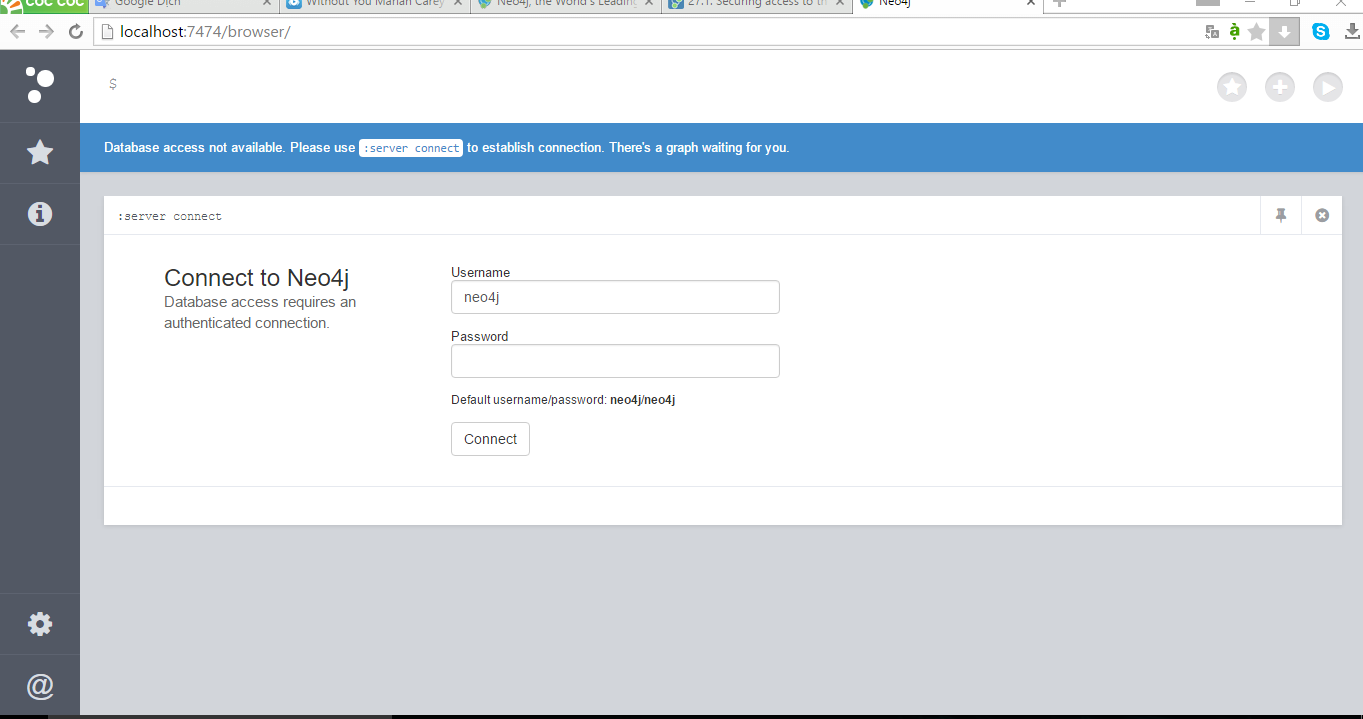
Trước hết bắt đầu với web interface của Neo4J để tìm hiểu làm thế nào Neo4J biểu diễn diễn dữ liệu dưới dạng đồ thị và làm thế nào đi đến, truy vấn đỉnh, cạnh xung quanh đồ thị.

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

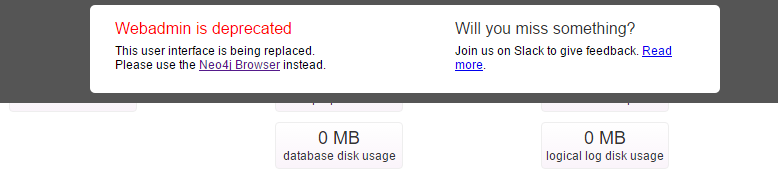
Sau đó chạy: <http://localhost:7474/browser/>



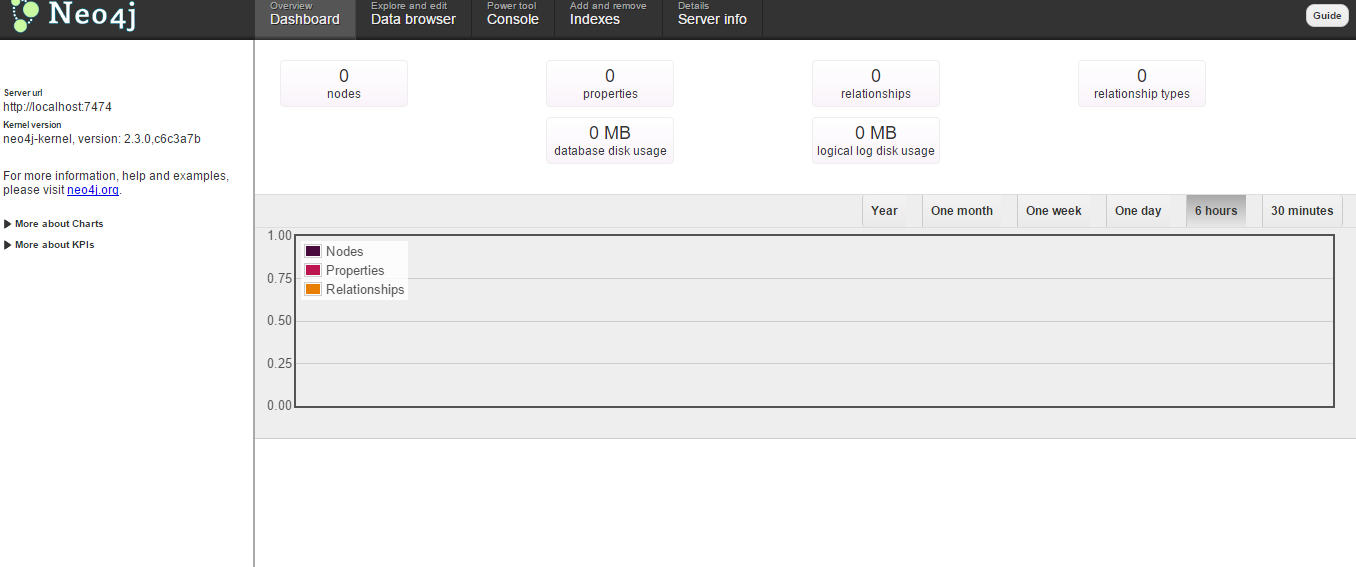
Nhập username và password. Bắt đầu tiến hành sử dụng

#### 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.

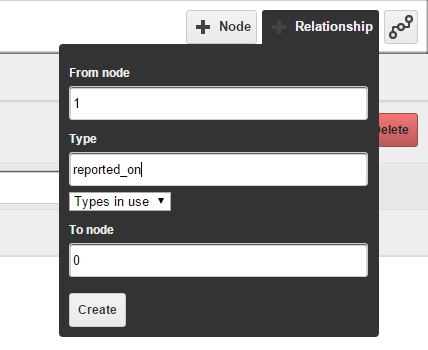
Lấy ví dụ về Wine đề cập ở phần trên.

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:



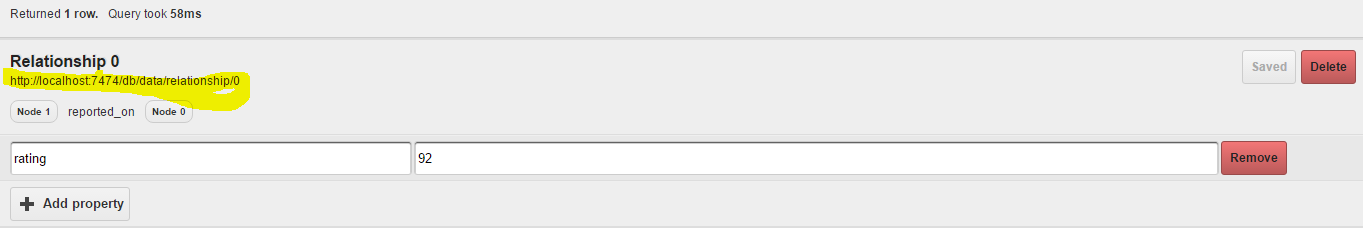
Tương tự ta cũng thêm một node với property là name và value là Wine Expert Monthly (có thể viết ngắn gọn như [name: “Wine Expert Monthly”]). Chú ý khi thêm thì chỉ số các node sẽ tự tăng (0, 1, 2, 3…n)

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.

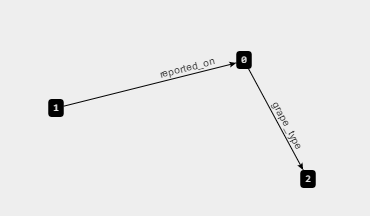


Sau khi biết loại rượu được tạo ra từ ai ta sẽ thêm một node để xác định loại nho nào.

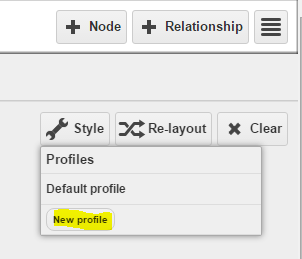
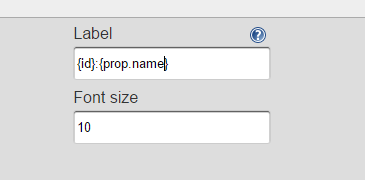
Add node riesling grape (nho reisling). Thêm property [name: “riesling”]. Sau đó tạo relationship từ 0 tới 2 với type là grape\_type và property [style: “ice wine”].

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

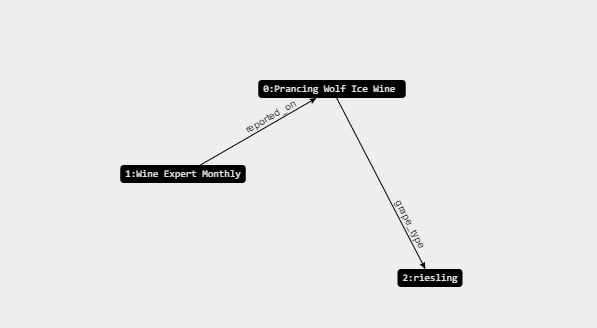
Được đồ thị như sau:



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

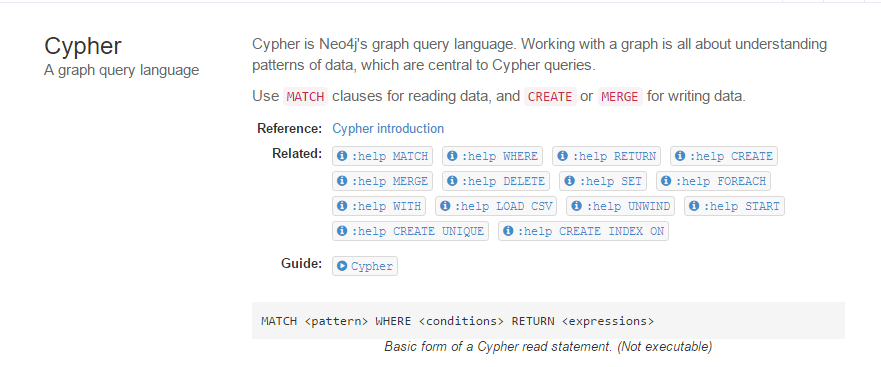
Điểu chỉnh label từ {id} sang {id}:{prop.name}. Nhấn save. Ta được kết quả như sau



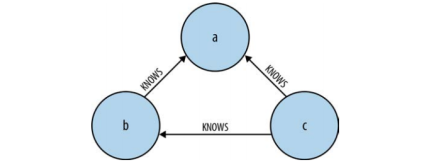
Với giao diện web interface dễ dàng thao tác thêm, xóa, sửa một node, relationship hỗ trợ cho việc làm việc với graph.

#### 2.2 Giới thiệu ngôn ngữ CYPHER và truy vấn đồ thị

Ở phần sau sẽ giới thiệu về ngôn ngữ Gremlin để truy vấn đồ thị, nhưng Gremlin hiện nay ít được sử dụng phổ biến trong truy vấn đồ thị Neo4J. Dùng phổ biến nhất là ngôn ngữ Cypher hỗ trợ trực tiếp trên trang chủ Neo4J vì vậy nên mình sẽ hướng dẫn sử dụng Cypher để dễ dàng tiếp cận:



Cypher là ngôn ngữ truy vấn cho Graph Database, có đặc điểm dễ đọc và dễ hiểu đối với các nhà phát triển, các chuyên gia cơ sở dữ liệu… Cypher cho phép người dùng tìm kiếm thông tin trên CSDL theo một mô hình cụ thể nào đó. Cypher được xem là ngôn ngữ truy vấn đồ thị dễ tìm hiểu nhất, và là một cơ sở dữ liệu tuyệt vời để học về đồ thị. Khi đã hiểu rõ về cypher, ta có thể dễ dàng học sang các ngôn ngữ truy vấn đồ thị khác.

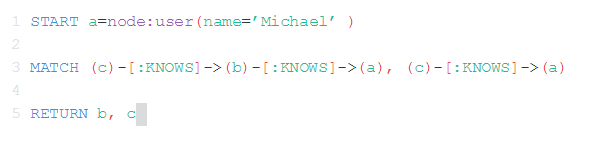


Đồ thị trên mô tả mối quan hệ bạn bè chung của ba người bạn. Ngôn ngữ Cypher sử dụng các mã ASCII để biểu diễn các mối quan hệ này như sau:



Mẫu trên trong ngôn ngữ Cypher mô tả một con đường kết nối từ c đến b, b đến a và từ c đến a. Nhìn chung, ngôn ngữ Cypher thể hiện rất tự nhiên từ cách chúng ta vẽ đồ thị trên “Whiteboard”.

Cũng như hầu hết các ngôn ngữ khác, Cypher cũng có các câu lệnh riêng. Câu truy vấn đơn giản nhất bao gồm một lệnh **START** theo sau bởi **MATCH** và **RETURN**. Sau đây là một ví dụ đơn giản cho câu truy vấn đơn giản sử dụng 3 mệnh đề trên để tìm bạn chung của người dùng có tên Michael:



* **Lệnh START:**

Lệnh START mô tả một hoặc nhiều điểm bắt đầu có thể là các nút hoặc các mối quan hệ trong đồ thị. Những điểm bắt đầu sẽ được chọn thông qua tìm kiếm chỉ mục hoặc tìm kiếm trực tiếp dựa vào các nút và các mối quan hệ. Như ví dụ phía trên, điểm bắt đầu sẽ là điểm mà có thuộc tính name và có giá trị là Michael. Giá trị trả về từ việc tìm kiếm điểm này là nút a (định danh a là do người dùng tự đặt). Định danh này được sử dụng trong suốt phần còn lại của câu truy vấn.

* **Lệnh MATCH:**

Đây là phần đặc tả bằng ví dụ từng phần. Ta sử dụng các kí tự ASCII để đại diện cho các nút và các mối quan hệ, “vẽ” ra các dữ liệu mà chúng ta quan tâm. Chúng ta sử dụng dấu đóng ngoặc và mở ngoặc để vẽ các nút (**Ví dụ**: **(a), (b), (c),…**), sử dụng cặp dấu gạch ngang kết hợp với dấu lớn hơn hoặc nhỏ hơn để vẽ ra các mối quan hệ (**–>, <–**). Các dấu lớn hơn (**>**) và nhỏ hơn (**<**) thể hiện hướng của quan hệ. Giữa cặp dấu gạch ngang có cặp đóng mở ngoặc vuông, nội dung trong cặp đóng mở ngoặc vuông được bắt đầu bằng dấu hai chấm và sau đó là tên của mối quan hệ **(Ví dụ: -[:KNOWS]->**).

Câu ví dụ trên miêu tả một đường dẫn gồm ba nút **(c)-[:KNOWS]->(b)-[:KNOWS]->(a), (c)-[:KNOWS]->(a)**, một trong số đó đã được nối với định danh a, những nút còn lại nối với b. Các nút này được được kết nối bởi các mối quan hệ **KNOWS**. Mô hình này về lý thuyết có thể được sử dụng nhiều lần trong dữ liệu đồ thị nên chúng ta nên ghim một phần chúng lại trong đồ thị.

Như vậy, chúng ta đã thực hiện xong việc tìm kiếm một nút thực sự trên đồ thị với mệnh đề**START** nút đại diện cho Michiael. Chúng ta gắn nút này vào định danh a, rồi đưa a sang mệnh đề **MATCH**. Từ đó ta đã ghim mô hình này đến một điểm trong đồ thị. Cypher khớp các phần còn lại của mô hình vào đồ thị vào xung quanh các điểm ghim. Từ đó thì nó tìm thấy các nút khác để gắn với các định danh phù hợp. Và như vậy, định danh a đại điện cho Michael, còn các định danh b, c sẽ theo định danh a mà gắn vào các nút xung quanh khớp với điều kiện trong mệnh đề MATCH.

* **Lệnh RETURN:**

Mệnh đề này quy định việc các nút, các mối quan hệ và các thuộc tính trong dữ liệu được nối nên được trả về như thế nào cho client. Trong ví dụ trên thì chúng ta mong muốn việc trả về là các nút kết nối với định danh b và c.

* **Một số lệnh khác của CYPHER:**

**WHERE**: cung cấp các tiêu chí để lọc các mẫu kết quả phù hợp.

**CREATE** và **CREAT UNIQUE**: tạo nút và các mối quan hệ.

**DELETE**: xóa nút, mối quan hệ và thuộc tính

**SET**: thiết lập các giá trị thuộc tính

**FOREACH**: biểu diễn một hành động cập nhật đối với mỗi phần tử trong một danh sách

**UNION**: hợp các kết quả từ 1 hoặc nhiều truy vấn

**WITH**: tạo chuỗi các phần truy vấn và chuyển tiếp các kết quả từ một tới kết quả tiếp theo

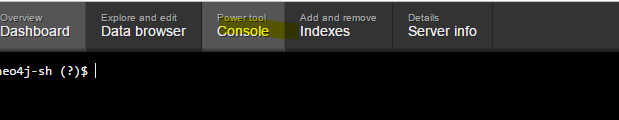
Cũng giống các ngôn ngữ truy vấn khác, Cypher còn nhiều mệnh đề khác như **SKIP**, **USING**, **MERGE**, **REMOVE**…

#### 2.3 Neo4j thông qua Gremlin

Trong phần này sẽ lấy ví dụ về bài toán liên quan đến rượu được giới thiệu ở mục 1.

Có rất nhiều ngôn ngữ tương thích với Neo4j như: Java code, REST, Cypher, Ruby 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, còn relationship 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]

Sẽ lấy ra được 3 đỉnh với node number lần lược là 1, 2 và 3.

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() method

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 muốn. 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:



#### 2.4 The Power of Pipes

Bạn có thể nghĩ Gremlin hoạt động như chuỗi 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 có 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).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.

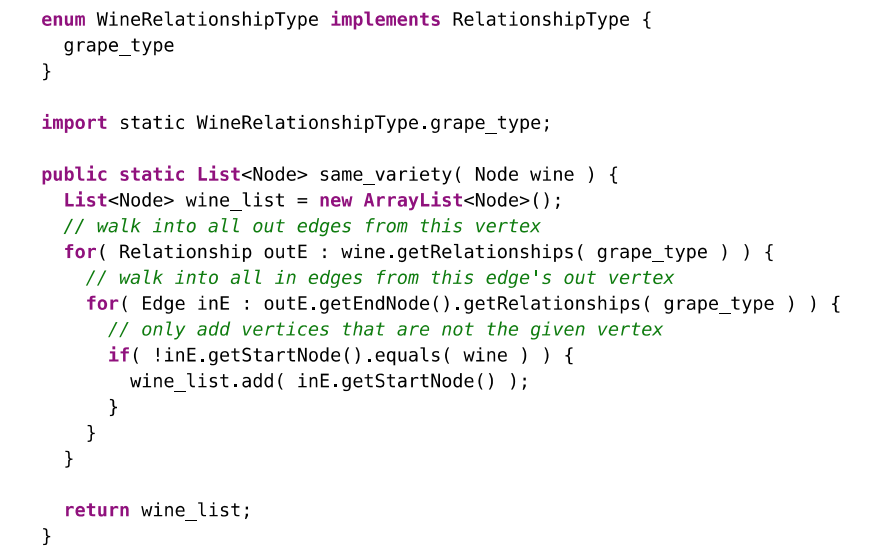
Gremlin là một ngôn ngữ xây dựng nhiều pipe. Đặc biệt nó xây dựng Java project tên Pipes. Để tìm hiểu khái niệm pipe, chúng ta trở lại wine graph. Giả sử bạn muốn tìm wines giống với rượu nhận được, chúng ta có các loại giống nhau. Bạn có theo dõi ice wine cũng có thể chia sẻ cạnh grape\_type với các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

Ice\_wine = g.v(0)

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

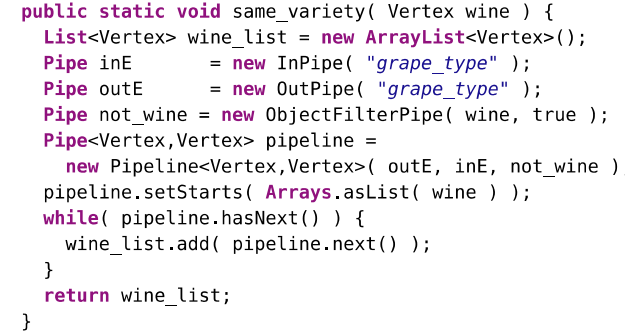
Từ đầu ra của ice\_wine là riesling ta sẽ tìm các đầu vào in có cạnh grape\_type trừ ice\_wine

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.



Project Pipes thiết kế một cách để khai báo các đỉnh vào (in) và ra (out). Bạn có thể tạo biểu đồ của in và out pipe, theo dấu và yêu cầu các giá trị từ pipeline. Sau khi lặp đi lặp lại việc sử dụng phương thức hasNext() của pipeline, nó sẽ trả về nút phù hợp tiếp theo. Trong cách khác, pipelin duyêt qua cây đồ thị cho bạn. Cho đến khi pipeline yêu cầu, bạn khai báo cơ bản làm thế nào cách đi đến các nút đó xảy ra.

Như minh họa, ở đây thực hiện phương thức same\_variety(). Sử dụng các phương thức nói trên và 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.

#### 2.5 Pipeline vs. Vertex

Để lấy một tập hợp chưa chỉ một đỉnh đặc biệt, bạn có thể theo vết nó từ một dãy các nút. Lấy ví dụ, g.V.filter {it.name == ‘reisling’}. V property là danh sách các nút, từ điều này chúng ta chọn lọc danh sách phụ. Nhưng khi bạn muốn đỉnh chính nó, bạn cần gọi phương thức 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 thuộc tính 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.

#### Schemaless Social

Neo4J hỗ trợ rất nhiều trong khía cạnh xã hội. Giúp thể hiện trực quan và quản lý các mối quan hệ và các vấn đề. Các công ty và tổ chức thường dùng để tư vấn, quản lý mạng xã hội (social network).Ví dụ tạo ra một khía cạnh xã hội từ graph dễ để thêm nhiều nút. 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’)

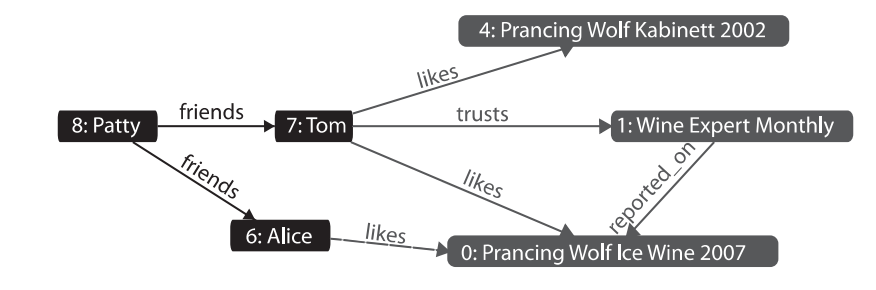
Patty là bạn của Tom và Alice nhưng là người mới về rượi, có một vài lựa chọn yêu thích.

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:



#### 2.7 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.

#### 2.8 Getting Groovy

Bên cạnh các bước thông qua Gremlin, chúng ta có thể dựa trên ngôn ngữ Groovy với các construct và method. Groovy có function map với tên collect() và một 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 nắm được ngôn ngữ lập trình Groovy trước đó. Một khi bạn nhận được bên trong sâu vào, bạn sẽ tìm thấy cách thực hiện các truy vấn mạnh mẽ của Neo4j.

#### 2.9 Domain-specific steps

Graph traversal cho phép duyệt qua các nút trong đồ thị thông qua một số quy tắc đảm bảo truy vấn với tốc độ cao, nhưng các doanh nghiệp và tổ chức có xu hướng ngược lại các ngôn ngữ domain-specific. Ví dụ, bạn không muốn hỏi thông thường như đỉnh nào với cạnh đến là grape\_type có chung cạnh đầu ra là đỉnh wine?

Gremlin là ngôn ngữ cụ thể để truy vấn miền 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



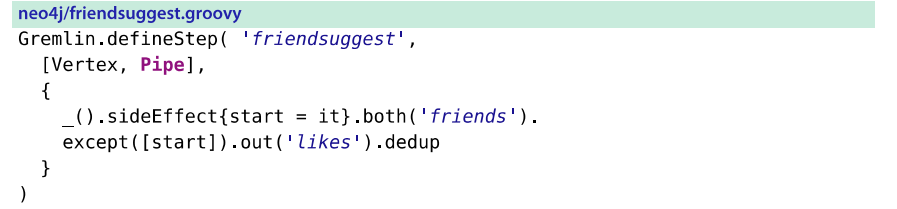
Đầu tiên chúng ta cho biết kỹ thuật Gremlin chúng ta thêm bước mới gọi là varietal. Dòng 2 cho Gremlin biết bước mới này phải đính kèm với Vertex và Pipe class. Dòng cuối là nơi điều kỳ diệu xảy ra. Hiệu quả tạo và mở {} chứa code thực thi. Bên dưới dấu gạch dưới và dấu ngoặc đại diện cho đối tượng pipeline. 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.

#### 2.10 Update, Delete, Done

Bạn có thể insert và thực hiện các bước thông qua đồ thị.

Về update và delete dữ liêu? Việc này dễ hơn nhiều, một khi bạn tìm thấy vertex hoặc edge bạn muốn sửa đổi. Hãy thêm mức độ 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 hư hại dữ liệu. Nhưng thường nó sẽ chỉ cảnh báo bạn lần tới kết nối vào đồ thị.

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

#### 3.1 Rest

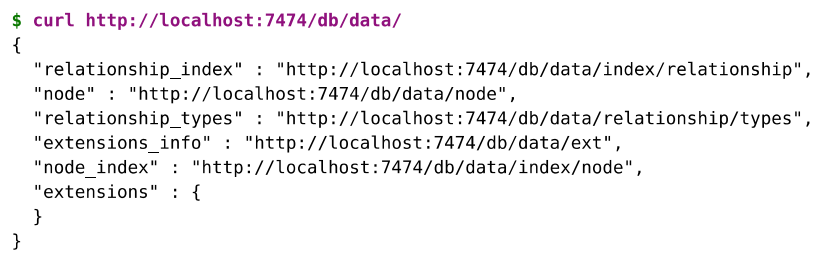
* Tổng quan: Rest là giao diện giúp người dùng giao tiếp với máy cơ sở dư liệu trong server một cách gián tiếp, có thể sử dụng để tạo node(nút), relationship(quan hệ giữa các nút) và thuộc tính của quan hệ. Sau đó dùng giao diện của Rest để đánh chỉ mục và thực hiện tìm kiếm các node mong muốn. Chúng ta có thể sử dụng một số ngôn ngữ đế sử dụng ré Gremlin query để truy vấn thông qua Rest API.
* Hiểu về giao diện Rest: Trong Neo4j, chúng ta có thể thao tác sử dụng Rest với ngon ngữ Gremlin. Bước đầu kiểm tra xem Rest server có đang chạy hay không (Figure 1)

Figure 1: Kết nối kiểm tra server bằng câu lệnh thông qua curl

* Sau khi start thành công server, kiểm tra thành công có thể sử dụng Rest để tạo các nút, quan hệ, thuộc tính tương tự như thao tác trên giao diện web đã nói ở phần trước đó. Gửi một phương thức post đến /db/data/node path với dữ liệu json(Figure 2) để tạo nút, tương tự bạn có thể tạo ra nhiều nút cho đồ thị.

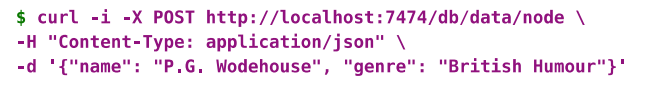


Figure : Tạo node cùng thuộc tính của node

Bên cạnh tạo nút, chúng ta có thể thêm thuộc tính cho nút khi tạo nút hoặc thêm sau khi tạo nút bằng cách gửi các gói tin phương thức Get.

* Thêm mối qua hệ giữa hai node thể hiện P. G. Wodehouse viết truyện ngắn “Jeeves Takes Charge”(Figure 3).

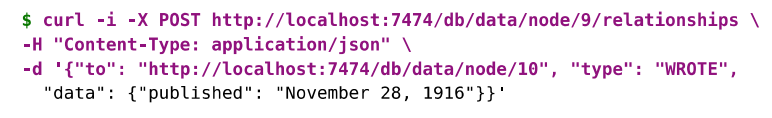


Figure : Thêm mối quan hệ giữa hai nút và thuộc tính

* 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(Figure 4)

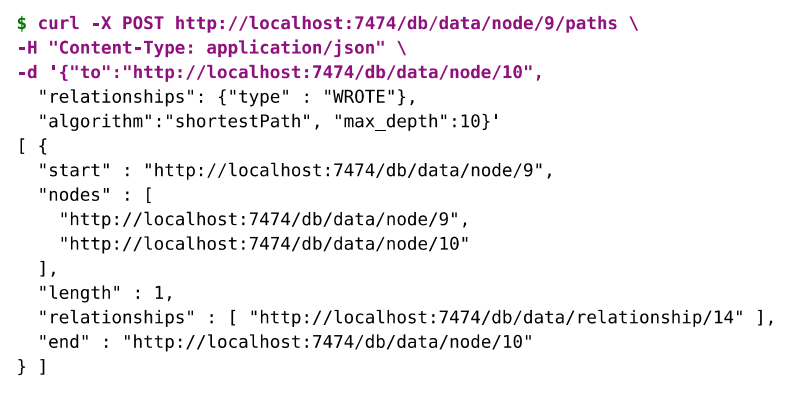


Figure : Tim đường ngắn nhất giữa hai nút bằng thuật toán ShortestPath

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

#### 3.2 Chỉ mục

* Tổng quan: Giống như nhiều cơ sở dữ liệu khách, NEO4J hỗ trợ đánh chỉ mục nhằm tăng hiệu suất khi tim kiếm giúp cho việc try vấn nhanh hơn. Chỉ mục đóng vai trò tương tự như mục lục của cuốn sách, chúng ta có thể đi đến nơi cần tìm bằng đường dẫn lưu trong chỉ mục mà không cần duyệt hết cả cuốn sách để tìm. 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.
* Đánh chỉ mục:

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 thêm tên indexes vào URL(Figure 5):

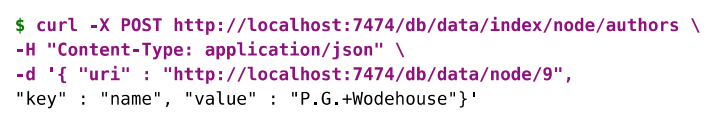


Figure : Tạo chỉ mục bằng key-value

Để lấy được dữ liệu chỉ đơn giản gọi index(Figure 6) là “name” đã đánh ở ví dụ trên:



Figure : Tìm kiếm đối tượng theo chỉ mục key-value

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(Figure 7):



Figure : Xây dựng chỉ mục fulltext

Thêm đối tượng có thên là P.G. +Wodehouse vào chỉ mục fulltext(Figure 8) để tìm kiếm sau này:

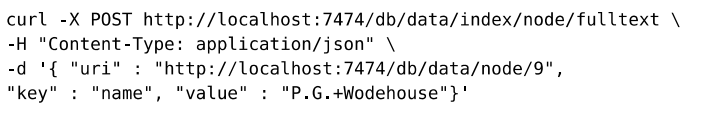


Figure : Thêm đối tượng vào fulltext

Tìm kiếm bằng thuật toán fulltext search theo cú pháp(Figure 9):



Figure : Tìm kiếm theo chỉ mục fulltext

Chỉ mục cũng có thể được xây dựng trên cạnh nối các nút, thay các nút bằng các mối quan hệ(relationship).

##### 3.2.2 BIG DATA:

Tìm hiểu về hiểu quả khi sử dụng NEO4J với big data. Xây dựng cơ sở dữ liệu với dữ liệu lớn với các nút, quan hệ, thuộc tính để tiến hành cài các thuật toán thông dụng. Ở đây chúng ta sử dụng cơ sở dữ liệu về điện ảnh từ trang web Freebase.com. Xây dựng và tiến hành đánh chỉ mục thuộc tính name để chạy thuật toán tìm kiếm. Dữ liệu có thể tồn tại số lượng lớn thông tin phim, nên phải cài đặt ruby hoặc JSON(Figure 9, Figure 10) để chạy script.

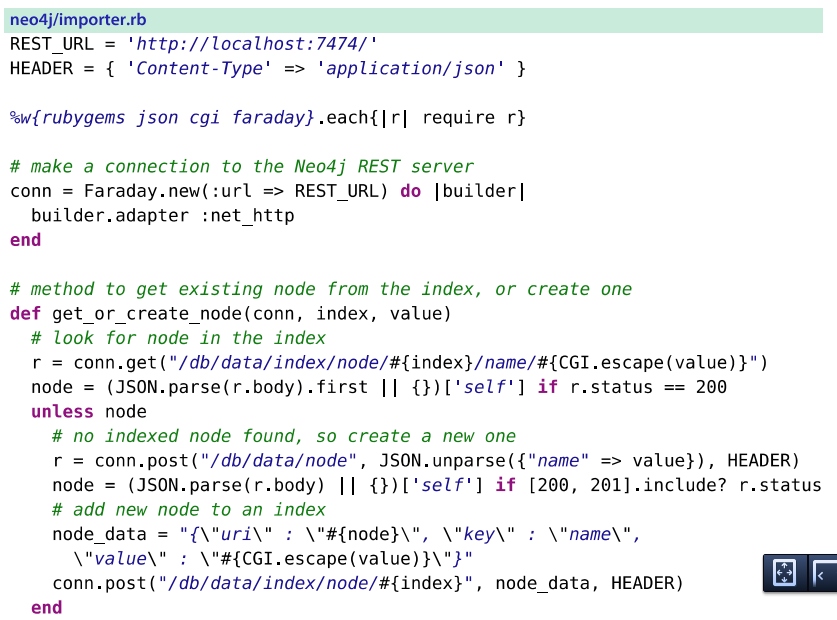


Figure : Thiết lập cài đặt cơ sở dữ liệu đồ thị cho noe4j

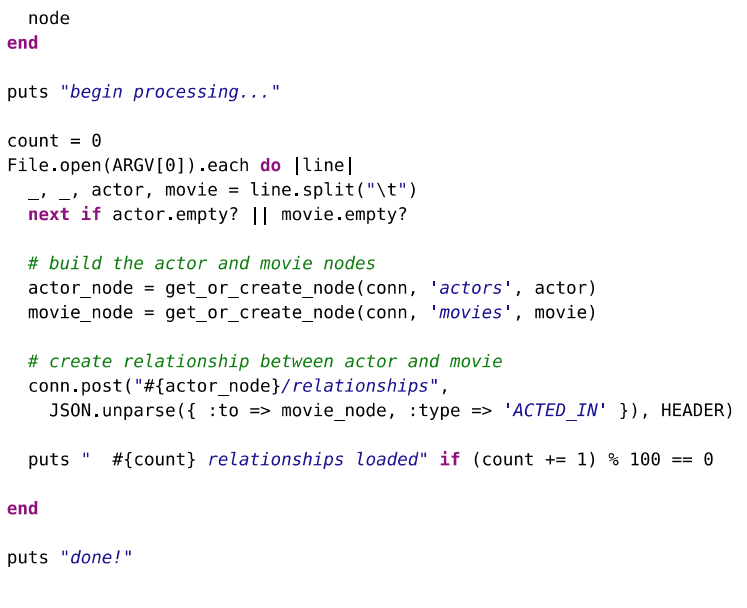


Figure : Thiết lập cài đặt cơ sở dữ liệu đồ thị cho noe4j

#### 3.3 Thuật toán thông dụng:

##### 3.3.1 Kevin Bakon:

Thuật toán Kenvin Bacon: Là thuật toán đồ thì nội tiếng trong việc tìm kiếm khoản đường đi ngắn nhất giữa hai nút. Lần đầu tiên được sử dụng để 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(Figure 11) để khởi tạo thuật toán:

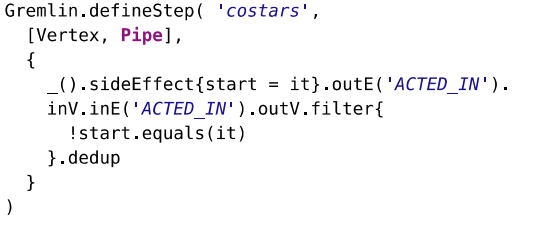


Figure : Thiết lập thuật toán Kevin Bakon

Trong NEO4J chúng ta không dùng nhiều truy vấn cho viết thiết lập dữ liệu. Điểm đặc biệt là từ nút đầu tiên chúng ta có thể tìm được nút gần nó nhất một cách dễ dàng bằng câu lệnh có cú pháp như hình bên dưới(Figure 12).

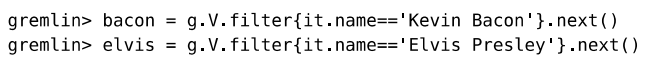


Figure : Thiết lập nút

Với ví dụ trên, giả sử đường đi của chúng ta có 6 nút, thông thường phải dùng 6 vòng lặp để tìm ra được con đường ngắn nhất giữa hai nút. Nhưng chúng ta đã chọn nút gần nó nhất nên số vòng lặp chỉ còn lại 4. Nói cách khác là vòng lặp xãy ra nếu chưa đặt 4 lần, chúng ta chưa tìm được nút bacon. Sau đó chúng ta xuất đường dẫn để đến được mỗi nut bacon tiếp theo(Figure 13).

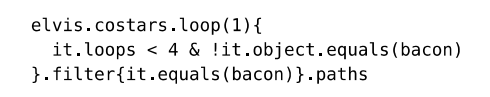


Figure : Thêm vòng điều kiện vòng lặp cho thuật toán Kevin Bakon

Dùng phương thức >> để liệt kê ra các nút trên đường đi sau khi chạy thuật toán(Figure 14).

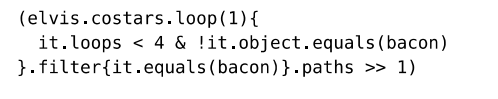


Figure : Thêm >> để lấy kết quả đường dẫn

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(Figure 15).

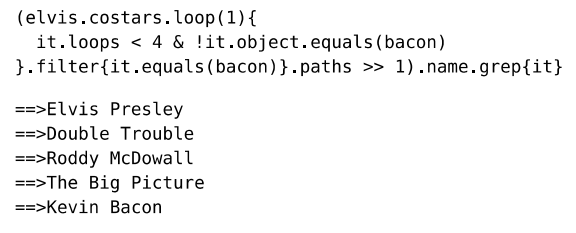


Figure : Kết quả tìm kiếm

##### 3.3.2 Thuật toán Random walk:

Thuât toán Random thích hợp với nhưng dữ liệu đồ thị lớn, khi số lượng các định quá nhiều, chúng ta không thể duyệt tất cả các đỉnh để tìm kiểm các nút gần một nút cho trươc bởi vì thời gian duyệt sẽ rất lâu. Thuật toán giúp tạo ra một số random và dùng con số này để lọc kết quả với một tỉ lệ mong muốn. Ví dụ: Bạn muốn tìm bạn diễn của Kenvin Bakon trong 1/3 của tổng số. Chúng ta sẽ tạo ra số random, khi lọc kết quả sẽ so sách số này với tỉ lệ cho trước, nếu thỏa mãn thì lấy kết quả nếu hợp lệ, không thỏa mãn thì tiếp tục lọc với nút khác chạy dòng lệnh rand = new Random(); để khởi tạo seed cho số 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 ). Cú pháp cài đặt (Figure 16)

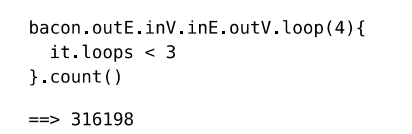


Figure : Cài đặt thuật toán Random Walk

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(Figure 17).

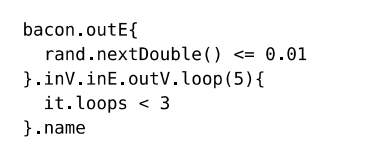


Figure : Lấy theo phần tram của một danh sách cho trước

##### 3.3.3 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(Figure 18).

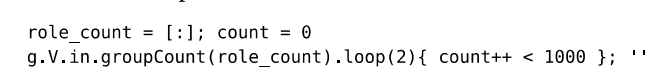


Figure : Thuật toán Centrality Park

Muốn sắp xếp kết quả trả về, 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.

##### 3.3.4 Thuật toán External Algorithms:

Chúng ta có thể tự tạo ra thuật toán để duyệt đỉnh trên cơ sở dữ liệu đồ thị, nhưng việc đó không cần thiết khi những thứ đó đã được xây dựng sẳn trong các bộ thư viện mã nguồn mở. Java Universal Network/Graph (JUNG) Framework la một tập hợp các thuật toán đồ thì phổ biến. Nhờ có dự án về 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.

Sau khi tả và cài đặt xong bằng gremlin console, đóng neo4j server và khởi động lại 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 chứa thuật toán tính điểm gọi là BarycenterScorer, thuật toán này sẽ 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(Figure 19).

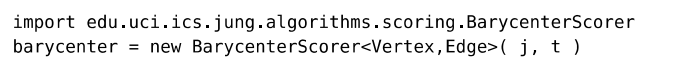


Figure : Import thuật toán vào GraphJung

Với nó, chúng ta có thể lấy điểm BarycenterScorer của nhiều node. Hãy tìm ra những điểm đó(Figure 20):

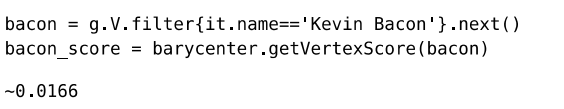


Figure : Lấy điểm BarycenterScorer

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 diễn được thêm vào(Figure 21):

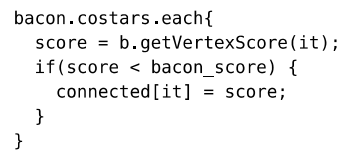


Figure : Duyệt đỉnh

Tất cả các value 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)

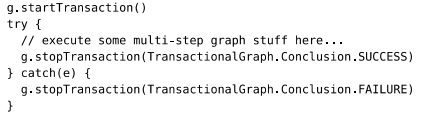
Neo4j là một cơ sơ dữ liệu giao tác ổn định, độc nhất, chắc, nguyên tử , giống PostgreSQL. Diều này làm cho nó lựa chọn tốt cho dữ liệu quan trọng, bạn có thể chọn một cơ sở dữ liệu quan hệ khác. Chí giống như các giao tác bạn đã thấy trước đó, giao tác Neo4j là những hoạt hoạt động tất cả hoặc không 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.

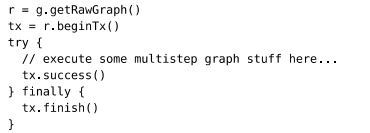
Giống như PostgreSQl, các dòng hàm cơ bản là tự động bao phủ một giao tác ẩn. Để giải thích giao tác đa dòng, chúng ta cần dố tượng cờ để tắt tự động phương thức giao tác, theo Neo4j chúng ta lên kế hoạch xử lý giao tác bằng tay. Ban có thể đổi phu87o7ng thức giao tác sang hàm *setTransactionMode()*



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*

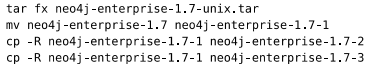
#### 4.2 High Availability

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

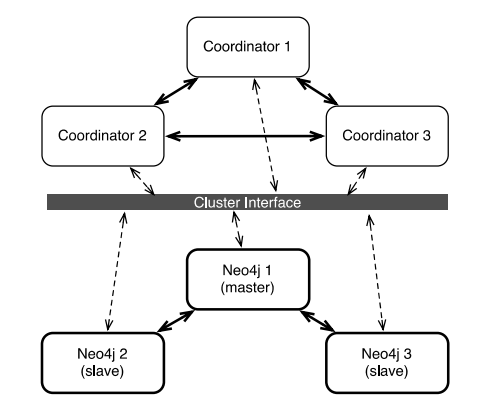
Để sử dụng Neo4j HA, chúng ta phải cài đặt một cluster. Neo4j dùng dich vụ điều phối cluster bên ngoài gọi là Zookeeper. Zookeeper là một đề án xuất sắc khác phát sinh từ dự án Apache Hadoop. Nó là một dịch vụ có mục đích chung để điều phối ứng dung. Neo4j HA sử dụng cái này để quản lý vòng đời hoạt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

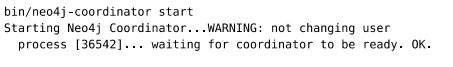


clientPort : 2181 cho server.1

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:

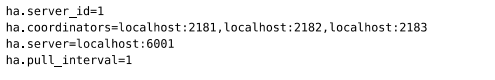


Thông thường port mặc định 7474 là ổn, nhưng khi chúng ta chạy 3 trường hợp Neo4j trên 1 hộp, chúng ta không thể khiến chúng lấn http/https. Chọn port 7471/7481 cho server.1, 7472/7482 cho server.2, 7473/7483 cho server.3

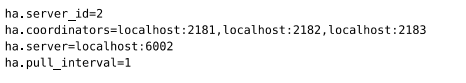


Port Neo4j server sẽ trao đổi thông tin vói nhau

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



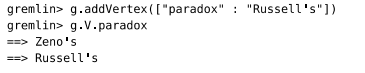
Kiểm tra server



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 .

1. Tài liệu tham khảo

* Tài liệu môn phát triển ứng dụng hệ thống thông tin hiện đại khoa CNTT – Đại Học Khoa Học Tự Nhiên.
* Tài liệu tại trang chủ Neo4J: <http://neo4j.com/> .
* <http://neo4j.com/docs/2.3.0/query-start.html>
* <https://zxthanhhuy.wordpress.com/category/graph-database/>