**Neo4J**

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 Whiteboard friendly**

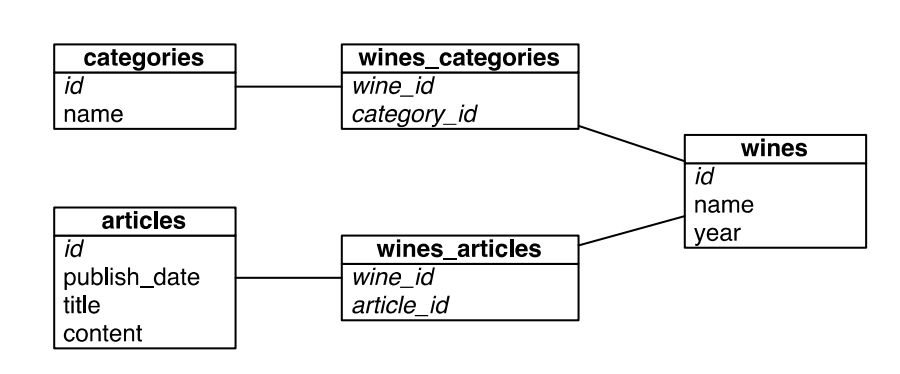
Tưởng tượng bạn phải tạo ra được lời đề nghị kỹ thuật rượu với nhiều thứ khác nhau như: Regions, wineries, vintages và designations.

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**

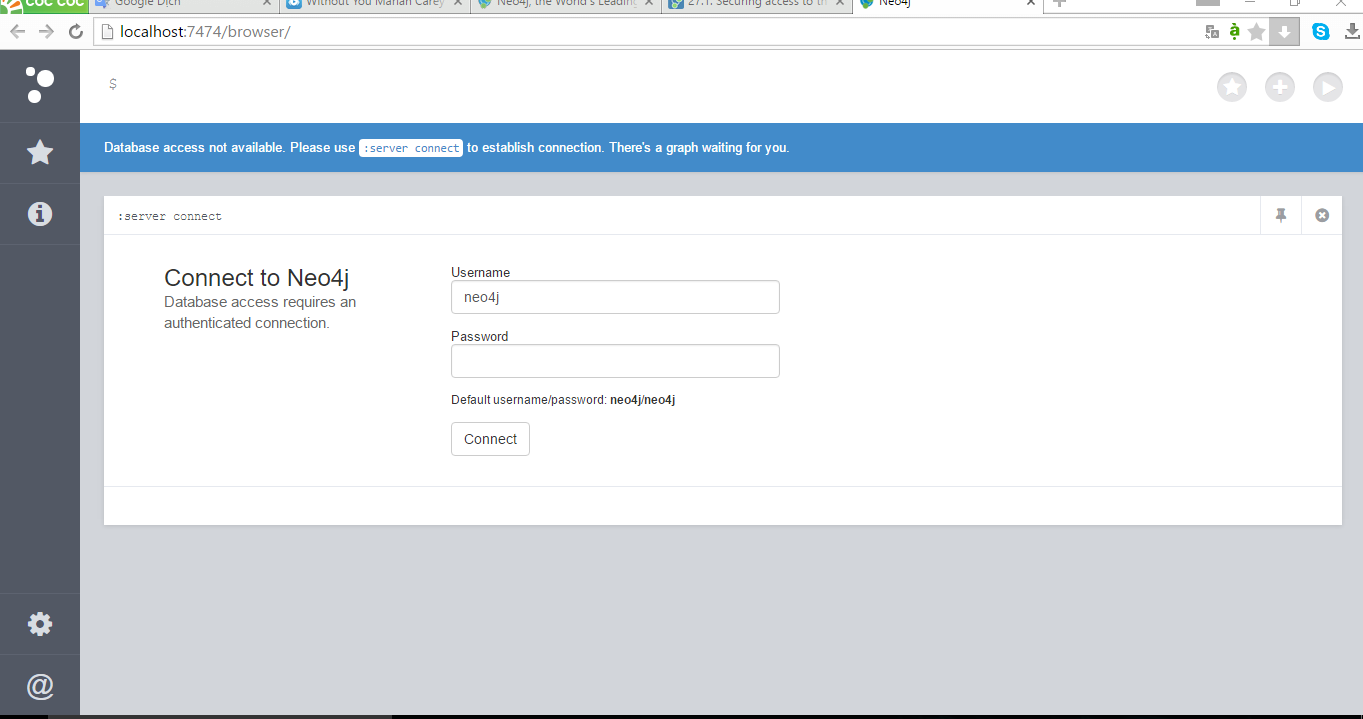
Trước hết bắt đầu với web interface của Neo4J để tìm hiểu làm thế nào Neo4J biểu diễn diễn dữ liệu dưới dạng đồ thị và làm thế nào đi đến, truy vấn đỉnh, cạnh xung quanh đồ thị.

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

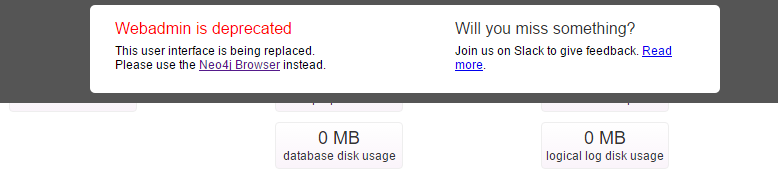
Sau đó chạy: <http://localhost:7474/browser/>



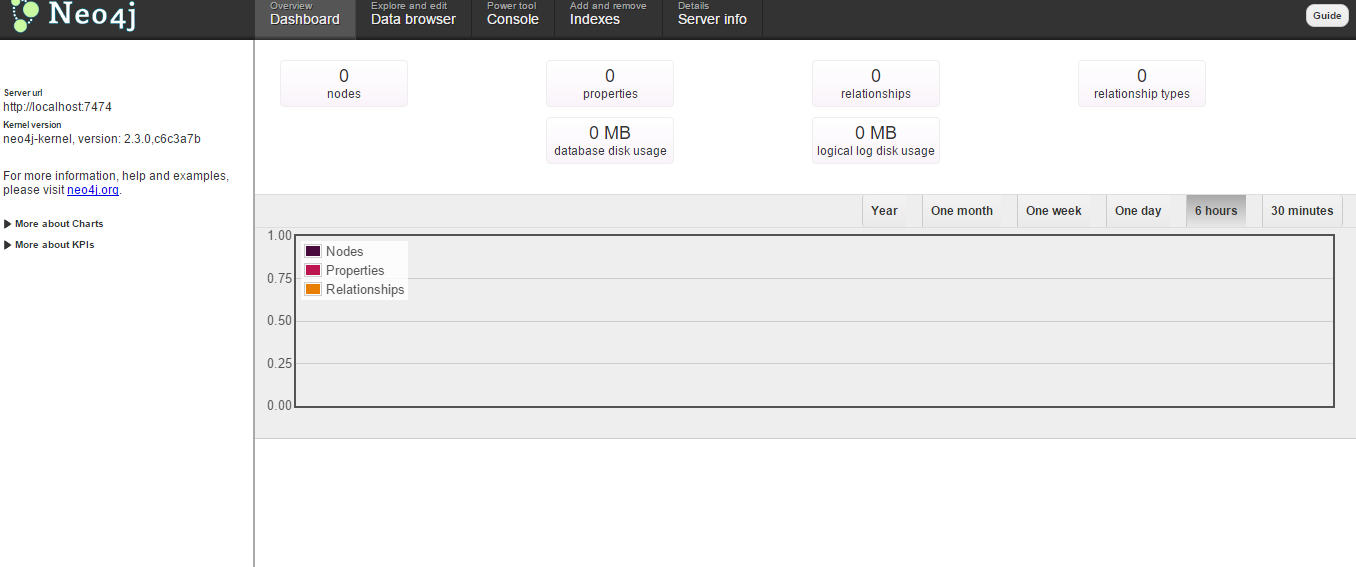
Nhập username và password. Bắt đầu tiến hành sử dụng

**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.

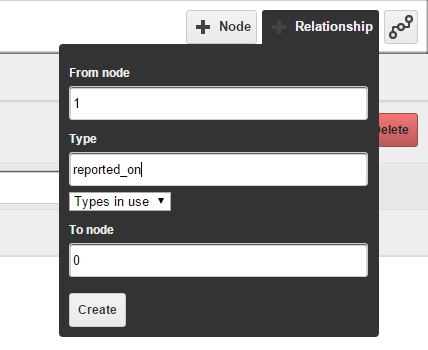
Lấy ví dụ về Wine đề cập ở phần trên.

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:



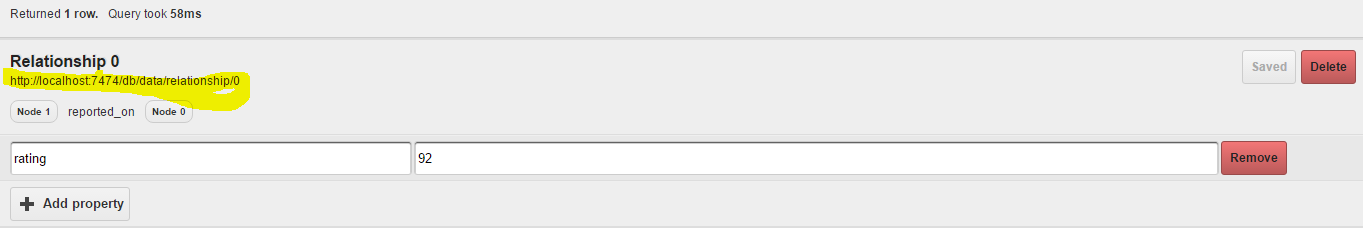
Tương tự ta cũng thêm một node với property là name và value là Wine Expert Monthly (có thể viết ngắn gọn như [name: “Wine Expert Monthly”]). Chú ý khi thêm thì chỉ số các node sẽ tự tăng (0, 1, 2, 3…n)

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.

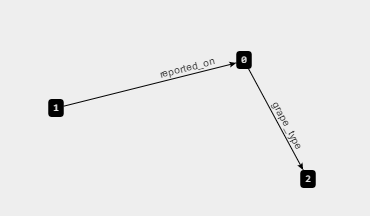


Sau khi biết loại rượu được tạo ra từ ai ta sẽ thêm một node để xác định loại nho nào.

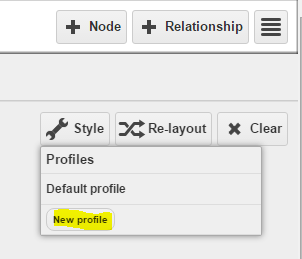
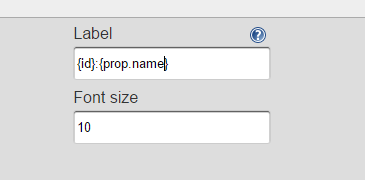
Add node riesling grape (nho reisling). Thêm property [name: “riesling”]. Sau đó tạo relationship từ 0 tới 2 với type là grape\_type và property [style: “ice wine”].

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

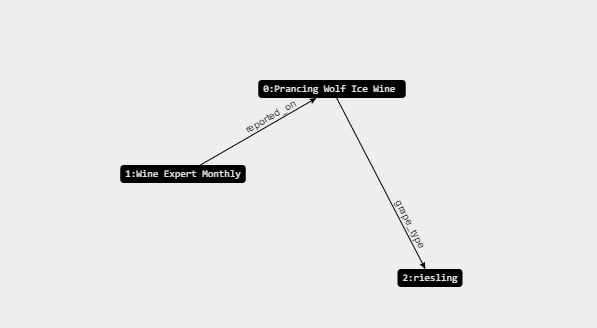
Được đồ thị như sau:



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

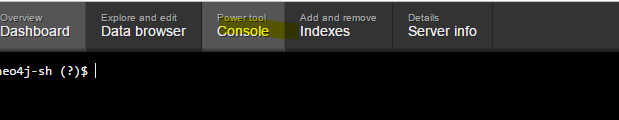


Với giao diện web interface dễ dàng thao tác thêm, xóa, sửa một node, relationship hỗ trợ cho việc làm việc với graph.

**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:



**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.

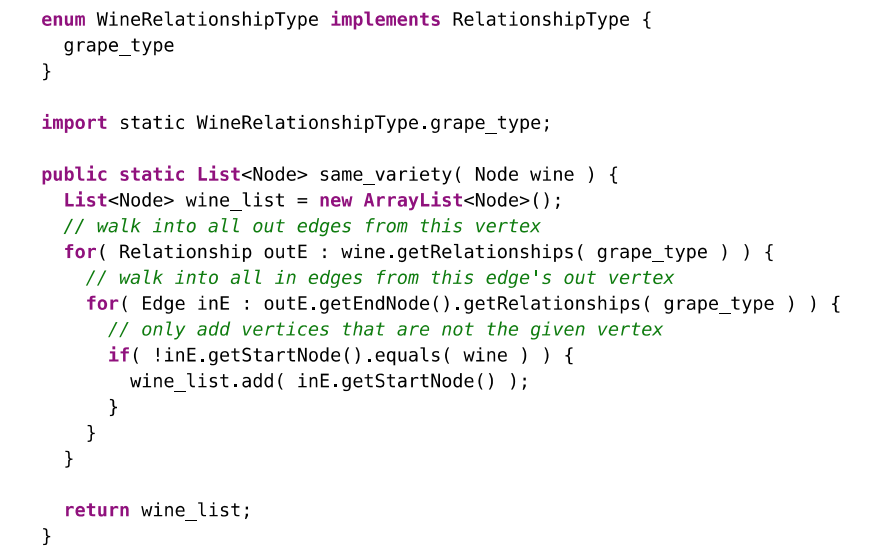
Gremlin là một ngôn ngữ xây dựng nhiều pipe. Đặc biệt nó xây dựng Java project tên Pipes. Để tìm hiểu khái niệm pipe, chúng ta trở lại wine graph. Giả sử bạn muốn tìm wines giống với rượu nhận được, chúng ta có các loại giống nhau. Bạn có theo dõi ice wine cũng có thể chia sẻ cạnh grape\_type với các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

Ice\_wine = g.v(0)

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

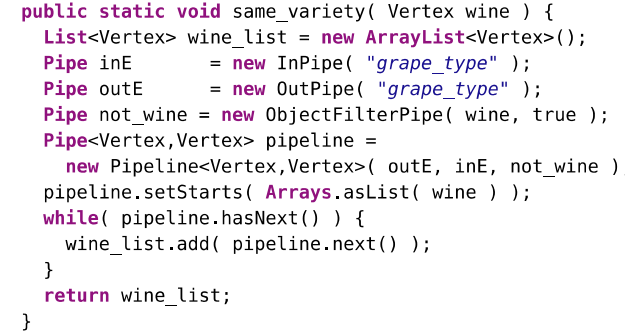
Từ đầu ra của ice\_wine là riesling ta sẽ tìm các đầu vào in có cạnh grape\_type trừ ice\_wine

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.

**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.

**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’)

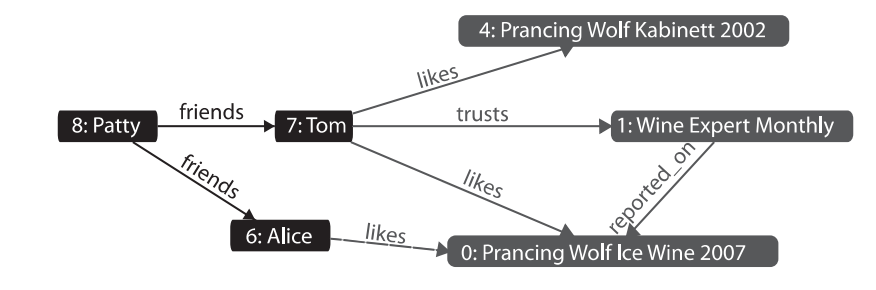
Patty là bạn của Tom và Alice nhưng là người mới về rượi, có một vài lựa chọn yêu thích.

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:



**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.

**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.

**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



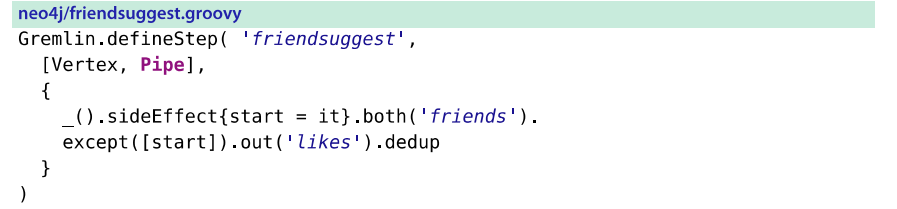
Đầu tiên chúng ta cho biết kỹ thuật Gremlin chúng ta thêm bước mới gọi là varietal. Dòng 2 cho Gremlin biết bước mới này phải đính kèm với Vertex và Pipe class. Dòng cuối là nơi điều kỳ diệu xảy ra. Hiệu quả tạo và mở {} chứa code thực thi. Bên dưới dấu gạch dưới và dấu ngoặc đại diện cho đối tượng pipeline.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

**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.

* **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/>