BÁO CÁO CUỐI KÌ NEO4J

1. Tổng Quan

Để nói đơn giản thì Graph Database-GD (cơ sở dữ liệu dạng đồ thị) là loại cơ sở dữ liệu được thiết kế để quản lý và xử lý những mối quan hệ của dữ liệu quan trọng như việc xử lý dữ liệu. Về cơ bản GD được thiết kế nhằm mục đích lưu trữ dữ liệu mà không cần đến việc khai báo hay định dạng trước mô hình của dữ liệu. Thay vào đó, dữ liệu được lưu trữ theo cách được ta vẽ trước – cho thấy sự liên hoặc kết giữa các mô hình thực thể.

a. Tại sao chọn Graph Database?

Mọi thứ xung quanh ta đều được kết nối với nhau. Không có bất kỳ dữ liệu nào là tồn tại một cách độc lập. Chỉ với cơ sở dữ liệu có khả năng lưu trữ và bao hàm dữ liệu ở mức thực tế nhất mới có thể xử lý, lưu trữ và truy xuất các mối quan hệ dữ liệu một cách hiệu quả. Những loại DB khác sử dụng phép “JOIN” trong mỗi lần kết nối các thực thể dữ liệu để truy vấn, Song GD lưu trữ những liên kết thực thể dữ liệu trong mô hình cùng chung với dữ liệu của mô hình.

Truy cập vào nodes(nút) và các mối quan hệ trong GD cho phép chúng ta duyệt qua các mối quan hệ từng dây từng phần tử một cách nhanh chóng, hiệu quả, trong 1 quãng thời gian thực.

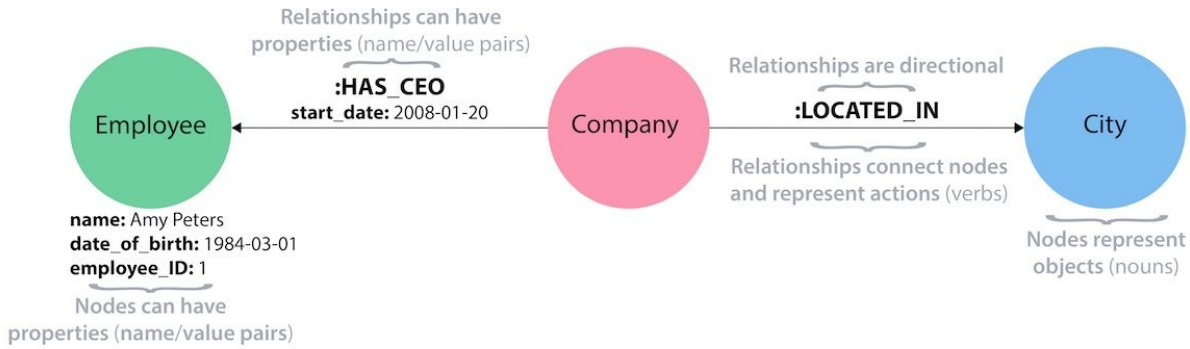
Không quan tâm đến kích thước của bộ dữ liệu(dataset), GD nổi trội trong việc quản lý dữ liệu có mối tương quan chặt chẽ và các truy vấn phức tạp. Chỉ với một cái mẫu và điểm bắt đầu, GD sẽ khám phá các dữ liệu kề cận với điểm xuất phát, thu thập và cộng gộp lại thông tin từ hàng triệu nodes và các mối quan hệ, không đụng tới những dữ liệu không liên quan tới chu vi tìm kiếm.

b. Mô hình Thuộc tính đồ thị (Property Graph Model)

Tương tự nhiều công nghệ, có một vài điểm tiếp cận riêng biệt tạo nên thành phần chính của GD. Một trong số đó là mô hình thuộc tính đồ thị, trong đó dữ liệu được tổ chức ở dạng nodes[nút], relationships[mối quan hệ], Properties[tình chất], (dữ liệu lưu trữ trên nodes hoặc relationships).

NODES: là các thực thể của đồ thị, không giới hạn số lượng thuộc tính(1 cặp key-value). Nodes có thể được gắn với labels, thể hiện các vai trò khác nhau trong miền dữ liệu. Labels của các nodes cũng có thể được sử dụng để gắn metadata cho các nốt cụ thể. (như là thông tin về index và các constraints)

Relationships (REL) cung cấp các sự kết nối được định hướng, định danh và các ngữ nghĩa liên quan giữa hai nodes thực thể (Ví dụ : 1 nhân viên LÀM VIỆC cho 1 công ty). Một relationship luôn phải có sự định hướng để duyệt, kiểu dữ liệu, có điểm đầu và điểm cuối. REL cũng có thể chứa property như các nodes và hầu hết các properties đều được định lượng. Như là về độ nặng, chi phí, khoảng cách, đánh giá ,thời gian, khoảng trung gian hoặc mức độ kết nối. Bởi việc lưu trữ dữ liệu 1 cách hiệu quả, số lượng hoặc các kiểu relationships có thể được dùng chung giữa 2 nodes mà không làm giảm hiệu năng giữa chúng. Cho dù chúng được lưu trữ ở các hướng khác nhau nhưng relationships vẫn luôn có thể được điều hướng theo cả 2 hướng của nodes



Nguồn ảnh: https://neo4j.com/developer/graph-database

c. Vậy NEO4J là gì?

NEO4J là một open-source, thuộc NOSQL, native graph database (csdl biểu dồ gốc) cung cấp ACID (atomicity, consistency, isolation, durability) tuân thủ theo transactional backend của ứng dụng của bạn.

NEO4J được biết đến như một native graph database bởi cách triển khai hiệu quả việc áp dụng mô hình thuộc tính đồ thị xuống đến tầng lưu trữ dữ liệu thay vì thông qua tầng xử lý business. Điều đó có nghĩa là nó được lưu chính xác theo bảng vẽ và csd; sẽ dùng con trỏ để định hướng và duyệt qua đồ thị. Trái với xử lý đồ thị hay các thư viện bộ nhớ trong, NEO4J còn cung cấp đầy đủ các đặc tính của cơ sở dữ liệu, bao gồm việc tuân thủ ACID trong transaction, hỗ trợ theo cụm và việc chuyển dự phòng trong thời gian runtime – khiến cho NEO4J thích hợp trong việc tạo đồ thị cho dữ liệu trong từng bối cảnh.

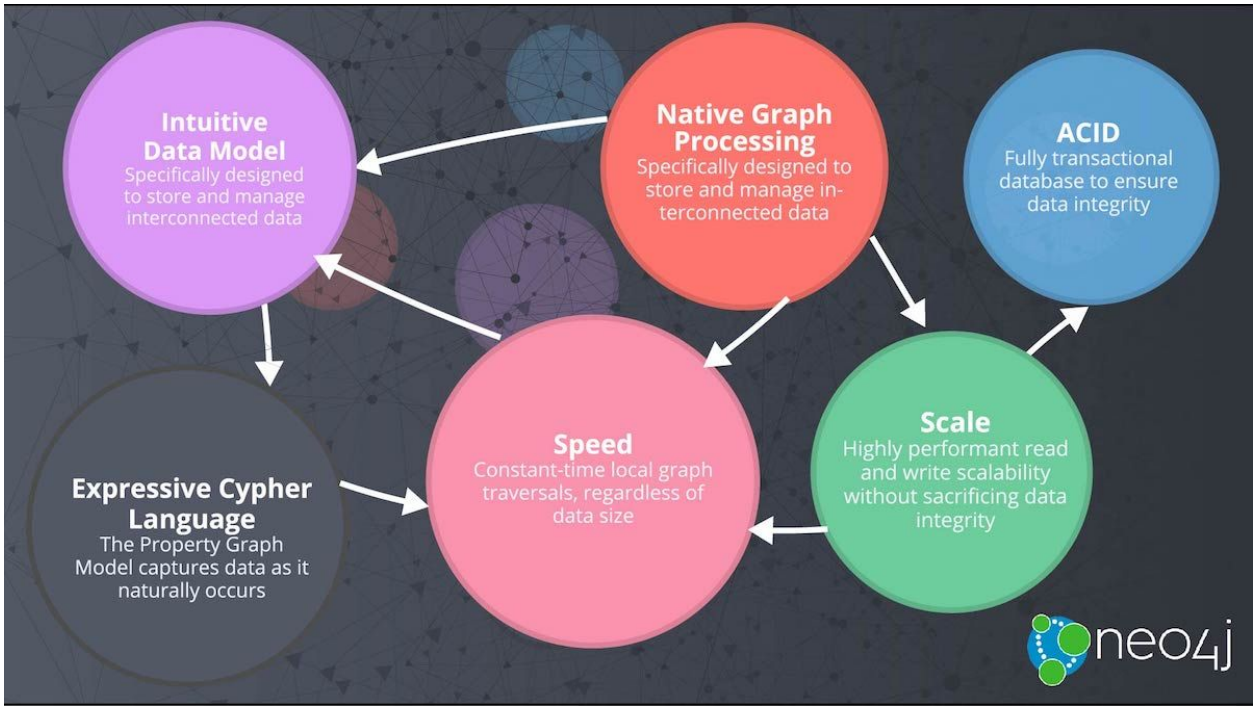
Một số tính chất rất hay được biết đến bởi các lập trình viên, kiến trúc sư và các nhà quản lý DB là:

- ✓ Cypher, một ngôn ngữ khai báo Query tương tự như SQL, nhưng tối giản hơn trong việc sử dụng đồ thị. Hiện tại đang được dùng là ngôn ngữ chung của một số database khác như Sap Hana Graph và Redis graph qua openCypher project (Dạng mở rộng của Redis [key-value]).

- ✓ Duyệt qua một đồ thị lớn với thời gian là 1 hằng số xác định nhờ vào sự lưu trữ hợp lý và sự diễn tả dữ liệu và các mối quan hệ 1 cách hiệu quả, có khả năng mở rộng tỷ lệ các nút lên đến hàng tỷ trên một phần cứng tương đối.

- ✓ Properties Graph có biểu đồ rất linh hoạt, có thể thích ứng theo thời gian. Việc này khiến biểu đồ có khả năng cụ thể hóa và thêm các relationship về sau để rút ngắn và cải thiện hiệu năng, tăng tốc cá miền dữ liệu khi tầng business cần thay đổi.

Drivers được cung cấp hỗ trợ các ngôn ngữ lập trình như JAVA, .NET, Python....



Nguồn ảnh: https://neo4j.com/developer/graph-database

2. So sánh cơ sở dữ liệu quan hệ với cơ sở dữ liệu dạng đồ thị:

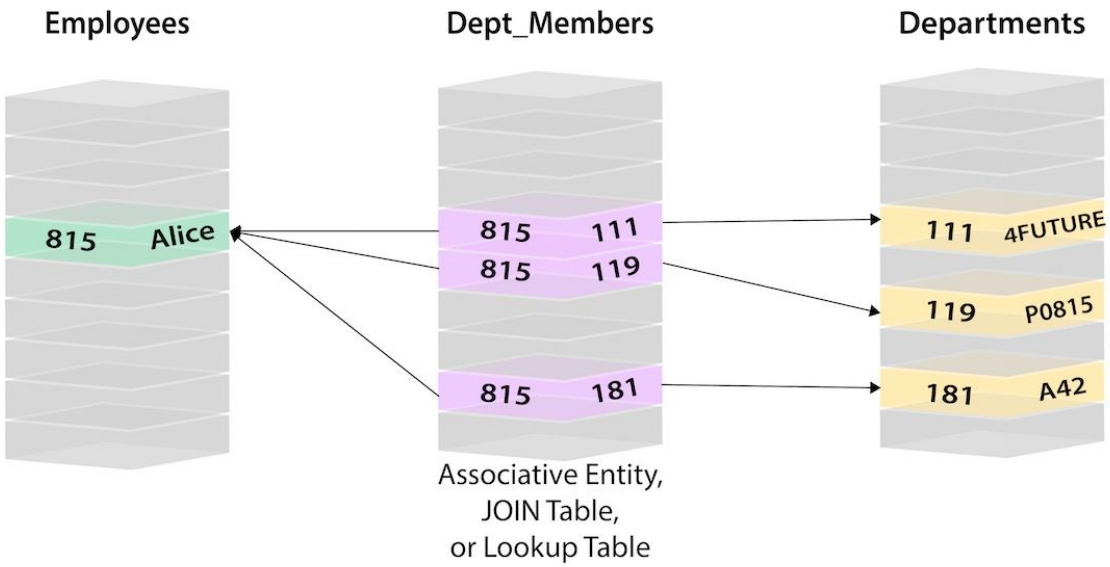
a. Cơ sở dữ liệu quan hệ

Lưu trữ dữ liệu có cấu trúc chặt chẽ trong những bảng với những cột có kiểu dữ liệu cụ thể và nhiều dòng định nghĩa kiểu thông tin trước. Nhưng do sự cứng ngắt trong cách tổ chức dữ liệu đòi hỏi nhà phát triển và ứng dụng phải được cấu trúc nghiêm ngặt theo.

Sự tham chiếu tới dòng và bảng khác được chỉ ra bởi thuộc tính khóa chính thông qua khóa ngoại. Các phép nối được tính toán tại thời điểm truy vấn bằng cách khớp các khóa chính và khóa ngoài của tất cả các hàng trong các bảng được kết nối. Các hoạt động này có tính toán phức tạp, yêu cầu bộ nhớ cao và có chi phí theo cấp số nhân.

Khi có nhiều mối quan hệ liên kết trong mô hình, ta phải tạo một bảng JOIN (hoặc bảng thực thể liên kết) chứa khóa ngoại của cả hai bảng, tăng thêm chi phí cho hành động kết bảng. Hình ảnh dưới đây cho thấy việc kết nối Person (từ bảng Person) đến một Department (trong bảng Department) bằng cách tạo một bảng kết Person-Department có chứa Person ID trong một cột và Department ID trong cột.

=> Điều này làm cho việc kết nối rất cồng kềnh bởi vì bạn phải biết Person ID và Department ID (thực hiện các tra cứu bổ sung để tìm chúng) để biết Person kết nối với các Department nào. Các lệnh kết tốn kém này thường được giải quyết bằng cách chuẩn hóa dữ liệu để giảm số lượng các phép nối cần thiết, do đó phá vỡ tính toàn vẹn dữ liệu của một cơ sở dữ liệu quan hệ.



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-rdbms/

b. Về kiểu đồ thị

Không giống như các hệ thống quản lý cơ sở dữ liệu khác, các mối quan hệ có tầm quan trọng tương đương trong mô hình dữ liệu đồ thị với chính dữ liệu đó. Nghĩa là không bắt buộc phải suy ra các kết nối giữa các thực thể sử dụng các thuộc tính đặc biệt như khóa ngoại hoặc tiến trình out-of-band như map-reduce.

Bằng cách tập hợp các nút và các mối quan hệ thành các cấu trúc được kết nối, cơ sở dữ liệu đồ thị cho phép chúng ta xây dựng các mô hình đơn giản và tinh vi để ánh xạ chặt chẽ với miền vấn đề của chúng ta. Dữ liệu ở đây khá giống với hình thức của nó trong thế giới thực - các thực thể nhỏ, chuẩn hóa, nhưng được kết nối rất phong phú. Điều này cho phép bạn truy vấn và xem dữ liệu của bạn từ bất kỳ điểm nào mà bạn có thể quan tâm hình dung được, hỗ trợ nhiều trường hợp sử dụng khác nhau.

Mỗi nút (thực thể hoặc thuộc tính) trong mô hình cơ sở dữ liệu đồ thị trực tiếp và vật lý chứa một danh sách các bản ghi mối quan hệ đại diện cho các mối quan hệ với các nút khác. Các bản ghi quan hệ này được sắp xếp theo loại, hướng và có thể chứa các thuộc tính bổ sung. Tương đương với một phép toán JOIN, cơ sở dữ liệu biểu đồ sử dụng danh sách này, truy cập trực tiếp vào các nút được kết nối và loại bỏ sự cần thiết phải tính toán tìm kiếm và so khớp tốn kém.

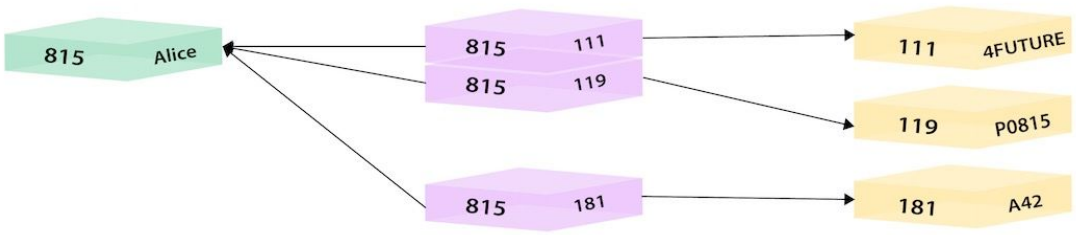
Khả năng này trước khi thực hiện các mối quan hệ vào cấu trúc cơ sở dữ liệu cho phép Neo4j cung cấp hiệu suất mạnh mẽ hơn, đặc biệt là cho các truy vấn phức tạp, cho phép người dùng tận dụng một phút đến mili giây.

c. Sự khác nhau giữa hai mô hình dữ liệu

Như bạn có thể thấy từ những khác biệt về cấu trúc được nêu ở trên, các mô hình dữ liệu quan hệ so với đồ thị rất khác nhau. Cấu trúc đồ thị đơn giản dẫn đến các mô hình dữ liệu đơn giản hơn và có tính "biểu cảm" cao hơn các mô hình được tạo ra bằng cách sử dụng các cơ sở dữ liệu quan hệ truyền thống hoặc NoSQL khác.

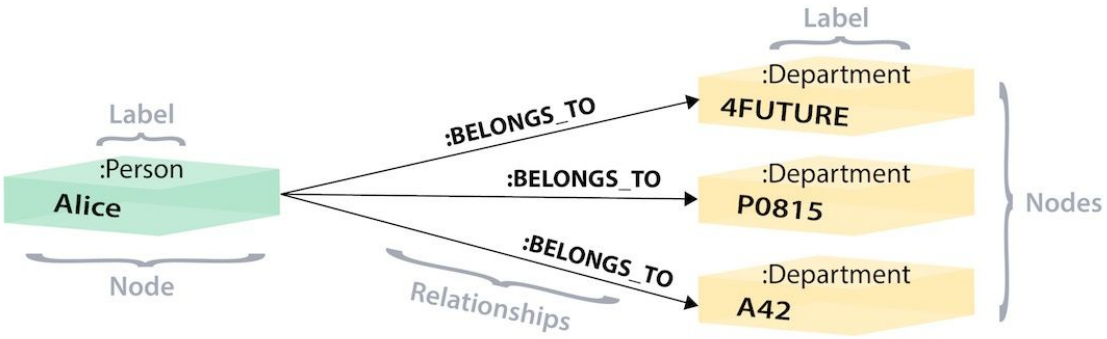
Nếu bạn quen với việc lập mô hình với các cơ sở dữ liệu quan hệ, hãy nhớ sự dễ dàng và vẻ đẹp của một sơ đồ quan hệ thực thể được chuẩn hóa tốt - một mô hình đơn giản, dễ hiểu một cách nhanh chóng. Biểu đồ chính xác là - một mô hình rõ ràng của miền, tập trung vào các trường hợp sử dụng mà bạn muốn.

➢ *Sau đây là ví dụ so sánh:*



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-rdbms/

Trong ví dụ quan hệ trên, chúng ta tìm kiếm bảng Person ở bên trái (có thể là hàng triệu hàng) để tìm user Alice và ID là 815. Sau đó, qua tìm kiếm bảng Person-Department (bảng giữa màu da cam) để định vị tất cả các hàng tham chiếu đến Person ID của Alice (815). Khi truy xuất 3 hàng có liên quan, chuyển đến bảng Department bên phải để tìm kiếm các giá trị thực của Department ID (111, 119, 181), ta biết Alice là một phần của 4Future, P0815 và A42.



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-rdbms/

Còn dạng đồ thị, ta có một nút cho Alice với Label của Person. Alice thuộc về 3 Department khác nhau, vì thế ta tạo ra một nút cho mỗi Department và với một Label của Department. Để biết Alice thuộc Department nào, ta sẽ tìm kiếm biểu đồ cho nút của Alice, sau đó duyệt qua tất cả các mối quan hệ BELONGS_TO từ Alice để tìm các nút Department mà Alice được kết nối. Một bước nhảy duy nhất không có liên quan đến tra cứu.

d. Lưu trữ và truy xuất dữ liệu

Việc truy vấn các cơ sở dữ liệu quan hệ rất dễ dàng với SQL - một ngôn ngữ truy vấn khai báo cho phép cả truy vấn ad-hoc dễ dàng trong một công cụ cơ sở dữ liệu, cũng

như truy vấn theo từng trường hợp cụ thể từ mã ứng dụng. Ngay cả những người lập bản đồ quan hệ đối tượng (ORM) cũng sử dụng SQL chuyên sâu để giao tiếp với cơ sở dữ liệu.

Ngôn ngữ truy vấn đồ thị khai báo của Cypher, Neo4j, được xây dựng trên các khái niệm cơ bản và các mệnh đề của SQL nhưng có rất nhiều chức năng cụ thể cho biểu đồ để giúp làm việc với mô hình biểu đồ dễ dàng hơn.

Nếu bạn đã từng viết một câu lệnh SQL với một số lượng lớn các phép nối, ta sẽ rối trong câu lệnh bởi cú pháp SQL. Trong Cypher, cú pháp vẫn ngắn gọn và tập trung vào các thành phần miền và các kết nối giữa chúng, thể hiện mẫu để tìm hoặc tạo dữ liệu trực quan và rõ ràng hơn.

Ví dụ truy vấn:

Trong SQL:

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Trong Cypher:

```
MATCH (p:Person)-[:WORKS_AT]->(d:Dept)
WHERE d.name = "IT Department"
RETURN p.name
```

- ✓ Áp dụng vào thực tế:

Sau đây là ví dụ truy vấn Department của John:

```
Connection con =
DriverManager.getConnection("jdbc:neo4j://localhost:7474/");

String query =
    "MATCH (:Person {name:{1}})-[:EMPLOYEE]-(d:Department) RETURN
d.name as dept";
try (PreparedStatement stmt = con.prepareStatement(QUERY)) {
    stmt.setString(1,"John");
    ResultSet rs = stmt.executeQuery();
    while(rs.next()) {
        String department = rs.getString("dept");
        ....
    }
}
```

Tóm tắt giữa CSDL quan hệ với CSDL đồ thị qua bảng so sánh như sau:

	CSDL quan hệ	CSDL đồ thị
Dữ liệu lưu trữ	Lưu trữ trong các bảng cố định, được xác định trước với các hàng và cột có dữ liệu được kết nối thường bị phân tách giữa các bảng, làm mất ổn định hiệu quả truy vấn.	Cấu trúc lưu trữ đồ thị với tính chất index-free liền kề dẫn đến các giao tác và xử lý nhanh hơn cho các mối quan hệ dữ liệu.
Mô hình hóa dữ liệu	Mô hình cơ sở dữ liệu phải được phát triển với người lập mô hình và được dịch từ một mô hình logic thành một mô hình vật lý. Vì các loại dữ liệu và các nguồn phải được biết trước, mọi thay đổi đều đòi hỏi mất nhiều thời gian ngừng hoạt động để triển khai.	Linh hoạt, mô hình dữ liệu dễ chỉnh sửa không có sự không khớp giữa mô hình logic và vật lý. Các kiểu dữ liệu và các nguồn có thể được thêm vào hoặc thay đổi bất cứ lúc nào, dẫn đến thời gian phát triển ngắn hơn đáng kể và bước lặp nhanh chóng thực sự.
Hiệu suất truy vấn	Hiệu suất xử lý dữ liệu bị ảnh hưởng bởi số lượng và độ sâu của phép JOIN (hoặc các mối quan hệ được truy vấn).	Xử lý đồ thị đảm bảo độ trễ bằng không và nhanh tức thì, bất kể số lượng hoặc độ sâu của các mối quan hệ.
Ngôn ngữ truy vấn (Query Language)	SQL: Một ngôn ngữ truy vấn làm tăng độ phức tạp với số lượng JOIN cần cho các truy vấn dữ liệu được kết nối.	Cypher: Một ngôn ngữ truy vấn biểu đồ gốc cung cấp cách hiệu quả nhất và có ý nghĩa nhất để mô tả các truy vấn quan hệ.
Hỗ trợ giao tác (Transaction Support)	Hỗ trợ giao tác ACID theo yêu cầu của các ứng dụng doanh nghiệp cho dữ liệu nhất quán và đáng tin cậy.	Giữ lại các giao dịch ACID cho dữ liệu thông suốt, đồng nhất và đáng tin cậy - hoàn hảo cho các ứng dụng doanh nghiệp toàn cầu luôn hoạt động.
Quy mô xử lý	Có thể scale thông qua nhân bản và mở rộng kiến trúc nhưng tốn kém. Mối quan hệ dữ liệu phức tạp không được thu hoạch theo quy mô.	Mô hình đồ thị vốn đã scale cho các truy vấn dựa trên mẫu. Quy mô kiến trúc duy trì tính toàn vẹn dữ liệu thông qua nhân bản. Các khả năng mở rộng quy mô lớn với các hệ thống Flash POWER8 và

		CAPI của IBM.
Hiệu suất trung tâm dữ liệu	Hợp nhất máy chủ là có thể nhưng tốn kém cho việc nâng cấp kiến trúc. Quy mô kiến trúc là tốn kém về mặt thuê mướn, năng lượng sử dụng và thời gian quản lý.	Dữ liệu và các mối quan hệ được lưu trữ một cách tự nhiên, cải thiện hiệu suất như sự phức tạp và quy mô phát triển. Việc hợp nhất máy chủ và sử dụng phần cứng cực kỳ hiệu quả hơn.

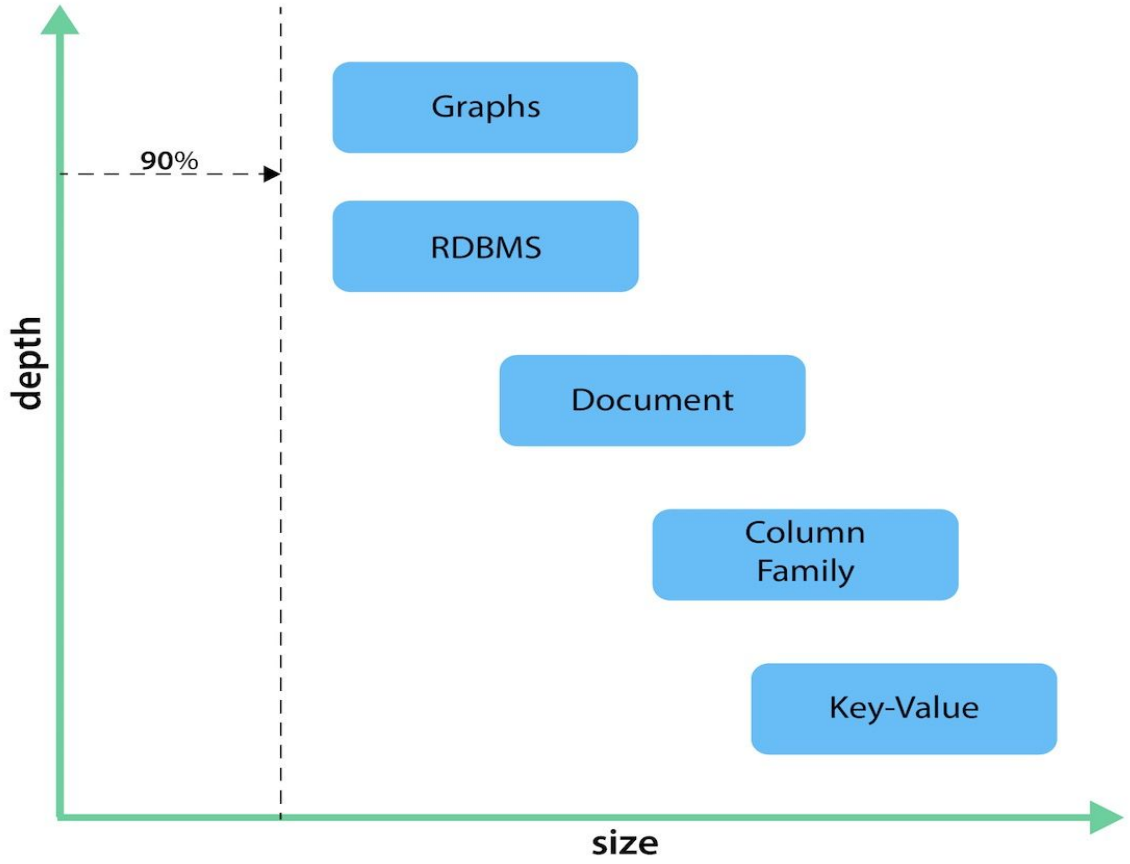
Nguồn: How Neo4j Compares - Neo4j vs. RDBMS
https://neo4j.com/product/#panel-compares-1

3. So sánh giữa NoSQL với Graph

Hầu hết các cơ sở dữ liệu NoSQL lưu trữ các thiết đặt của các tập bị ngắt kết nối. Điều này gây khó khăn trong việc sử dụng chúng cho dữ liệu và đồ thị được kết nối. Một chiến lược phổ biến cho việc thêm mối quan hệ vào các lưu trữ như vậy là nhúng tập các số nhận dạng (identifier) bên trong trường thuộc một tập hợp khác - mang lại hiệu quả cho các khóa ngoại. Tuy nhiên, điều này đòi hỏi phải tham gia tập hợp ở cấp ứng dụng - nhanh chóng trở nên cực kỳ tốn kém.
— Graph Databases, O'Reilly

Các cơ sở dữ liệu NoSQL khác thiếu các mối quan hệ. Mặt khác, cơ sở dữ liệu đồ thị xử lý các mạng thông tin chi tiết, cung cấp bất kỳ cái nhìn về dữ liệu của bạn phù hợp với trường hợp sử dụng của bạn. Đảm bảo giao tác phổ biến và đáng tin cậy từ các hệ thống quan hệ cũng như bảo vệ các cập nhật dữ liệu biểu đồ trong Neo4j, tuân theo các tiêu chuẩn ACID.

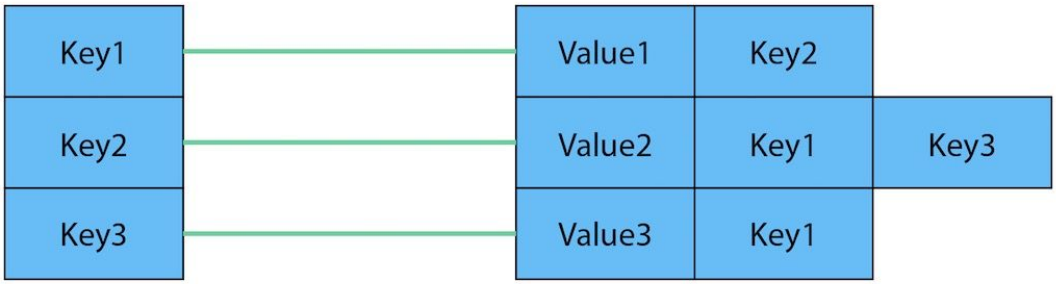
Sau đây chi tiết so sánh giữa NoSQL với graph:



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-nosql/
Chiều sâu của mô hình CSDL

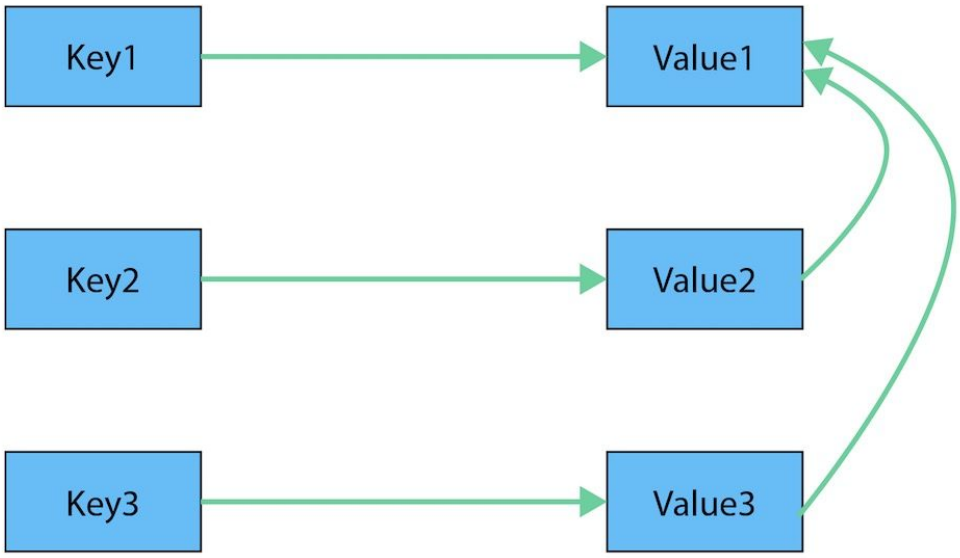
a. Lưu trữ key-value (khóa-giá trị) và đồ thị:

Mô hình key-value có hiệu suất cao để tra cứu số lượng lớn các giá trị đơn giản hoặc thậm chí phức tạp. Hình ảnh dưới đây cho thấy cách lưu trữ key-value điển hình được cấu trúc.



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-nosql/

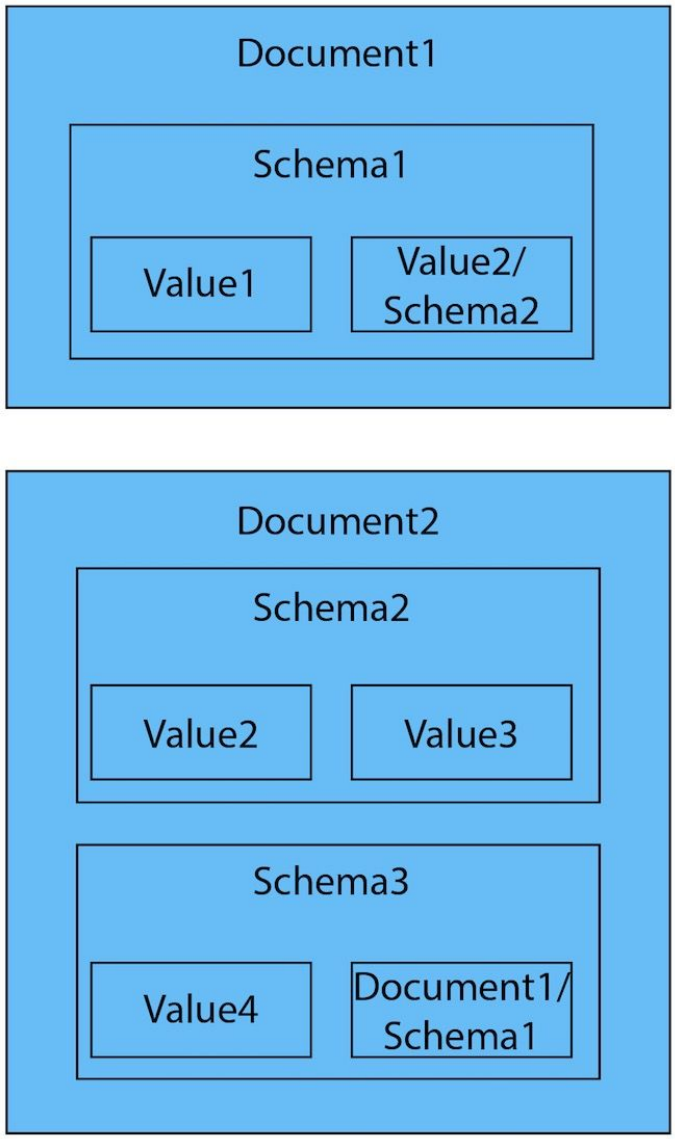
Tuy nhiên, khi các giá trị được kết nối với nhau, bạn có biểu đồ. Neo4j cho phép bạn duyệt nhanh giữa các giá trị được kết nối và tìm thông tin chi tiết trong các mối quan hệ. Phiên bản đồ thị dưới đây cho thấy mỗi khóa liên quan đến một giá trị như thế nào và các giá trị khác nhau có thể liên quan như thế nào với nhau (giống như các nút được kết nối với nhau thông qua các mối quan hệ).



Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-nosql/

b. Lưu trữ văn bản (Document) với đồ thị:

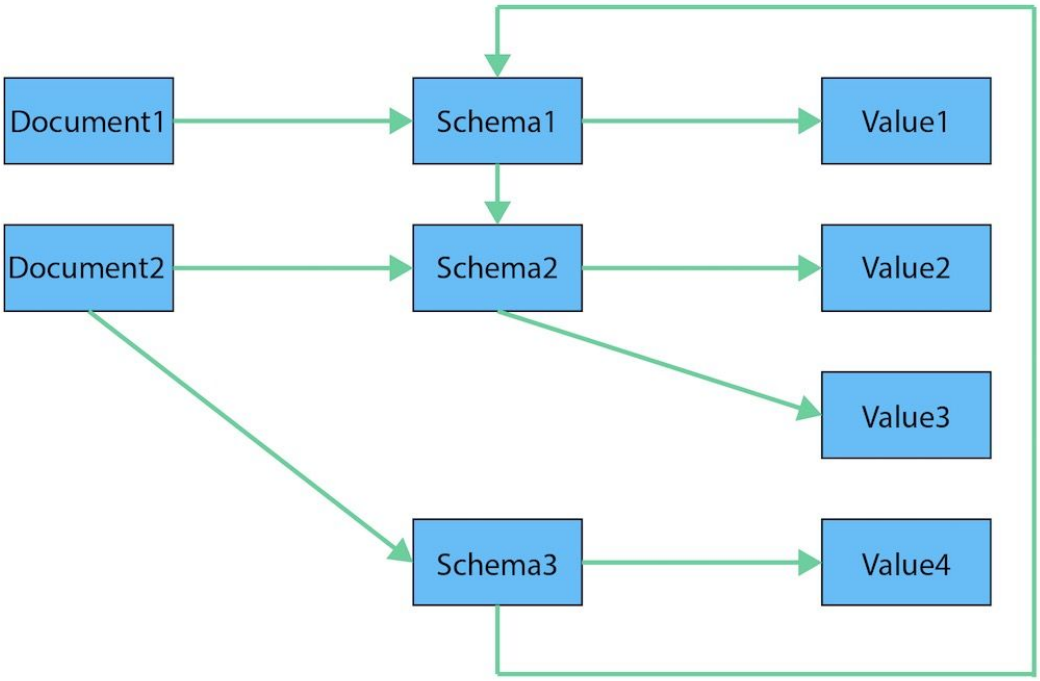
Phân cấp có cấu trúc của mô hình Document chứa rất nhiều dữ liệu không có lược đồ có thể dễ dàng được biểu diễn dưới dạng cây. Mặc dù cây là một loại đồ thị, một cây chỉ thể hiện một phép chiếu hoặc phối cảnh của dữ liệu. Hình ảnh dưới đây minh họa cách cấu trúc phân cấp lưu trữ Document dưới dạng các phần trong các thành phần lớn hơn.



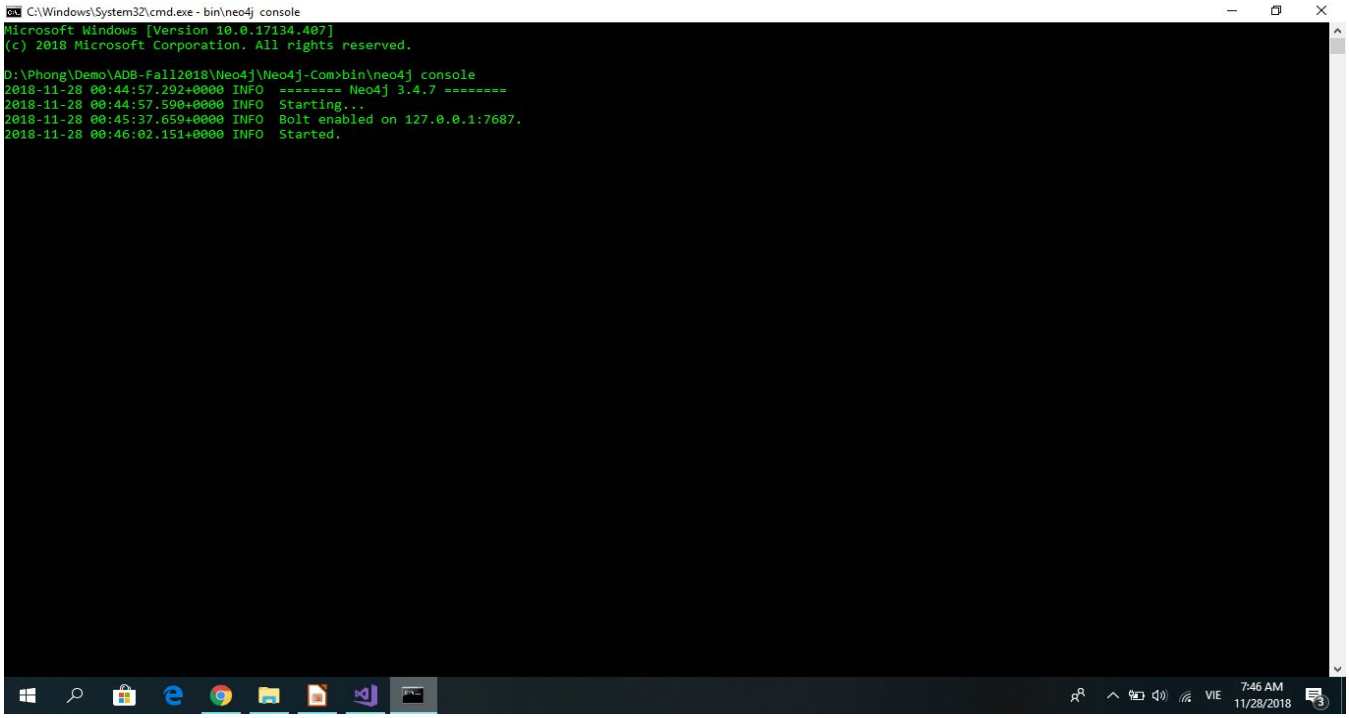
Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-nosql/

Nếu bạn liên kết các Document khác (hoặc chứa các phần tử) trong cây đó, bạn có một biểu diễn rõ ràng hơn về cùng một dữ liệu và có thể dễ dàng điều hướng bằng cách sử dụng biểu đồ. Mô hình dữ liệu đồ thị cho phép nhiều hơn một biểu diễn tự nhiên xuất

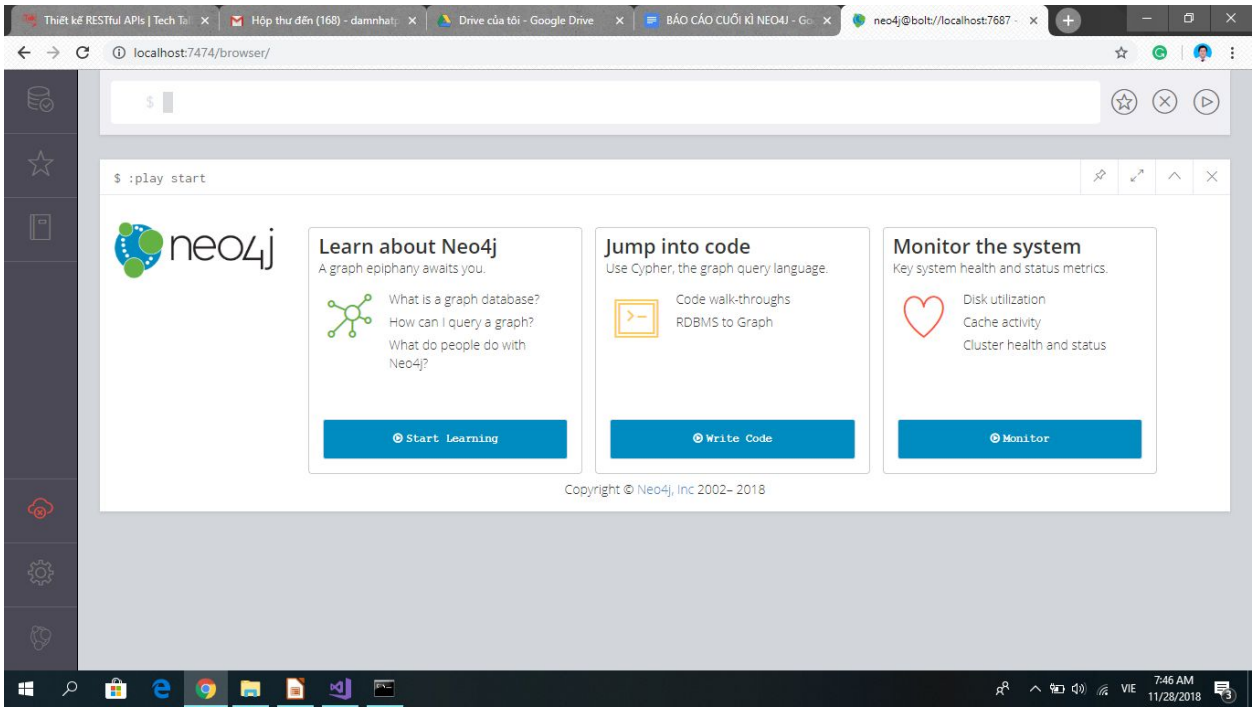
hiện động khi cần. Phiên bản đồ thị dưới đây cho thấy cách di chuyển dữ liệu này đến cấu trúc biểu đồ cho phép bạn xem các cấp độ và chi tiết khác nhau của cây trong các kết hợp khác nhau.



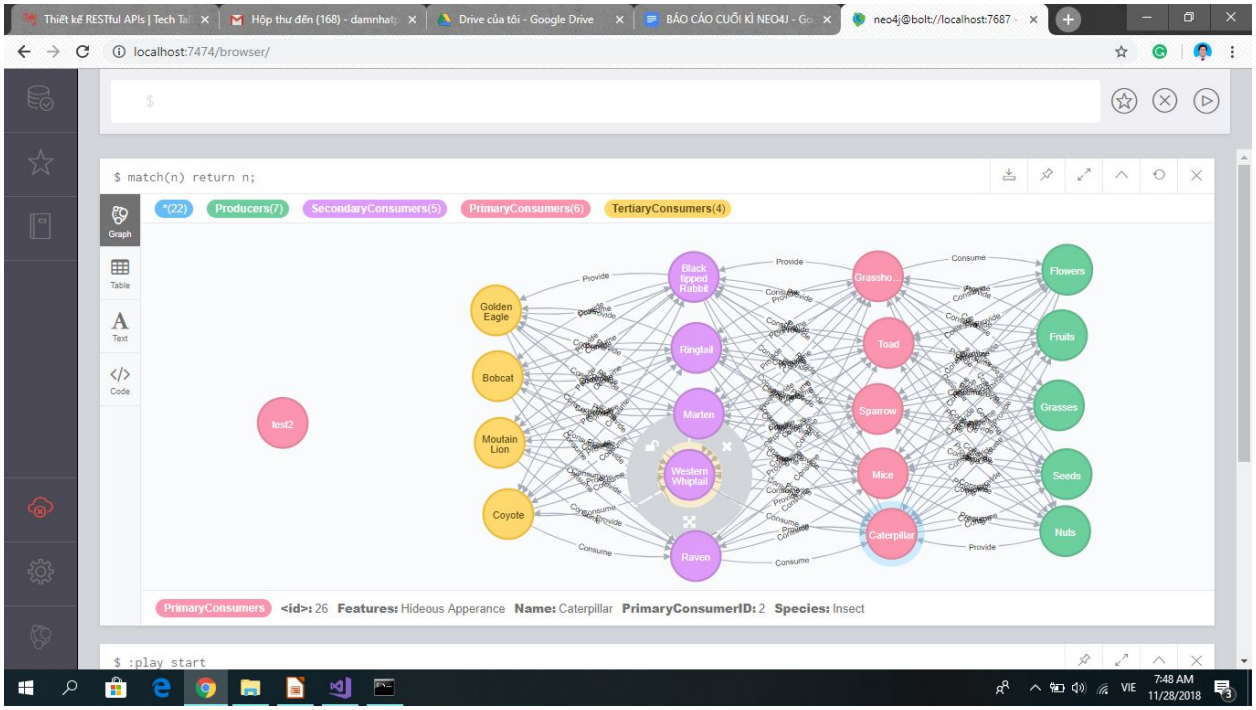
Nguồn ảnh: https://neo4j.com/developer/graph-db-vs-nosql/



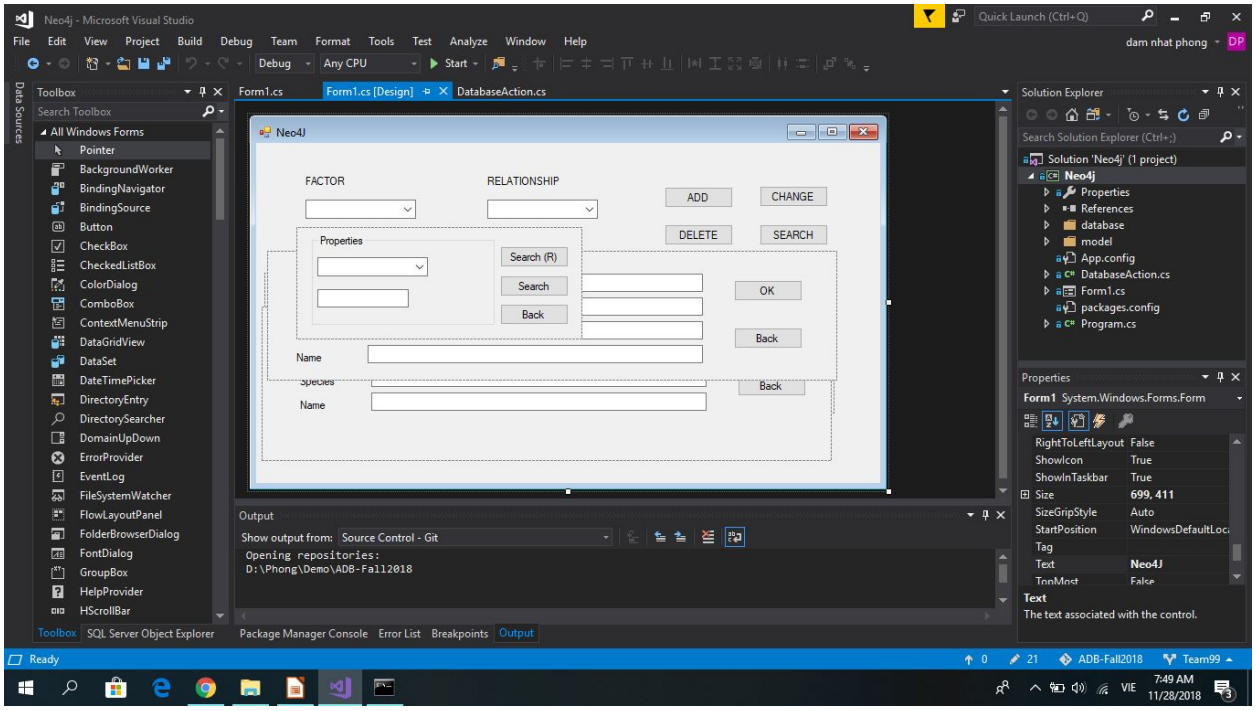
Cách khởi động neo4j



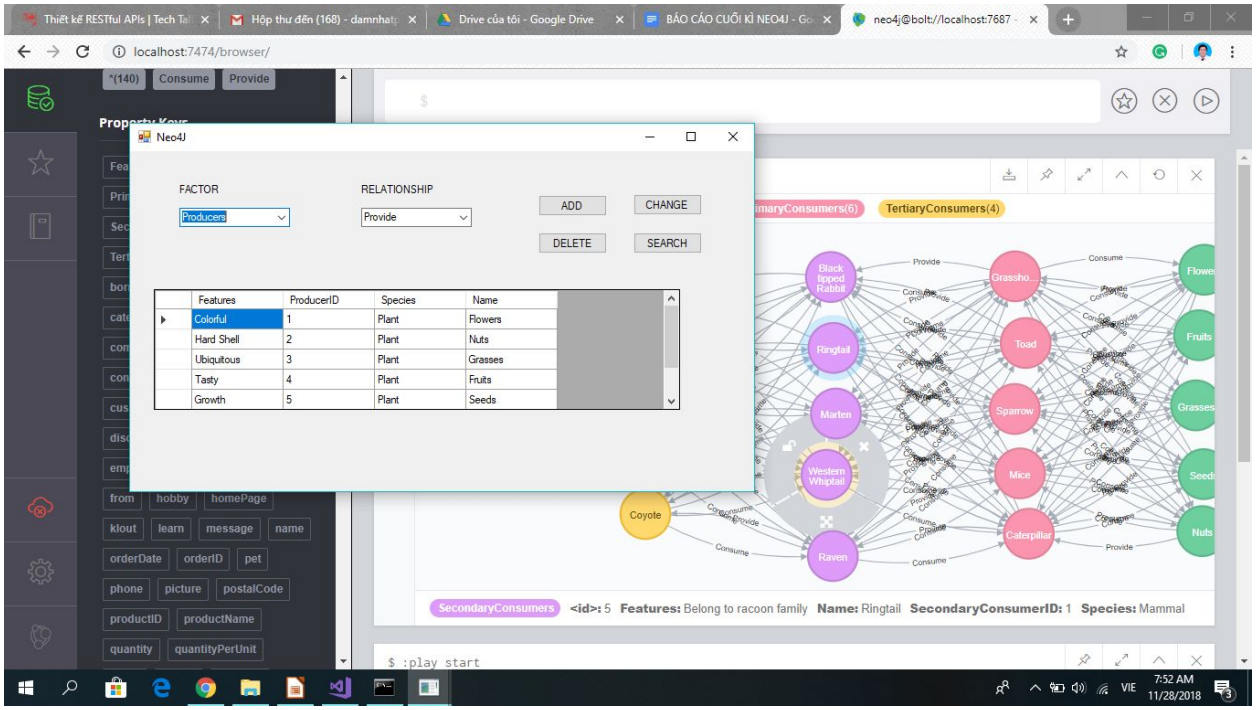
Giao diện neo4j khi truy cập vào http://localhost:7474



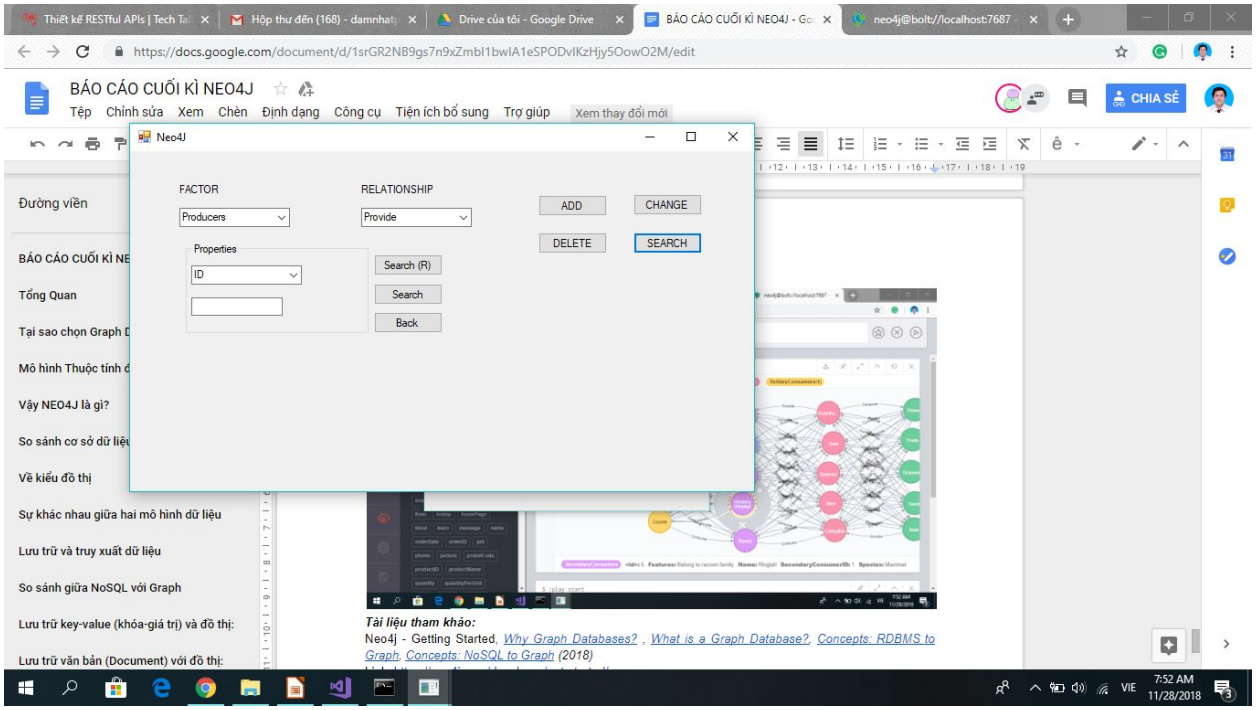
Cơ sở dữ liệu "hệ sinh thái"



Thiết kế giao diện demo



Chạy chuong trình demo



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Neo4jProject.model
{
    class Neo4jUser
    {
        public string username { get; set; } = "neo4j";
        public string password { get; set; } = "password";
    }
}
```

Username và password để đăng nhập vào neo4j

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Neo4jProject.model
{
    class PrimaryConsumers
    {
        public int PrimaryConsumerID { get; set; }
        public string Name { get; set; }
        public string Species { get; set; }
        public string Features { get; set; }
    }
}
```

Khởi tạo các thuộc tính

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Neo4jProject.model
8  {
9      class Producers
10     {
11         public int ProducerID { get; set; }
12         public string Name { get; set; }
13         public string Species { get; set; }
14         public string Features { get; set; }
15     }
16 }
17
```

```
7  namespace Neo4jProject.model
8  {
9      class SecondaryConsumers
10     {
11         public int SecondaryConsumerID { get; set; }
12         public string Name { get; set; }
13         public string Species { get; set; }
14         public string Features { get; set; }
15     }
16 }
17
```

```
7  namespace Neo4jProject.model
8  {
9      class TertiaryConsumers
10     {
11         public int TertiaryConsumerID { get; set; }
12         public string Name { get; set; }
13         public string Species { get; set; }
14         public string Features { get; set; }
15     }
16 }
17
```

```
namespace Neo4jProject.database
{
    class DatabaseExecution:IDisposable
    {

        private readonly IDriver _driver;

        public DatabaseExecution(string uri, string username, string password)
        {
            _driver = GraphDatabase.Driver(uri, AuthTokens.Basic(username, password));

        }
```

Kết nối cơ sở dữ liệu

```
namespace Neo4j
{
    class DatabaseAction:IDisposable
    {
        private readonly IDriver _driver;
        public DatabaseAction(IDriver driver) {
            _driver = driver;
        }

        public List<string> GetAllFactorLabels() {
            List<string> list = new List<string>();
            using (var session = _driver.Session())
            {
                var result = session.WriteTransaction(t =>
                    t.Run("match(n) return distinct labels(n) as label").ToList());
                foreach (var tem in result)
                {
                    string[] str;
                    str = JsonConvert.DeserializeObject<string[]>(JsonConvert.SerializeObject(tem[0]));
                    list.Add(str[0]);
                }
            }

            return list;
```

```
public List<string> GetFactorRelationShipByLabel (string label)
{
    List<string> list = new List<string>();
    using (var session = _driver.Session())
    {
        var result = session.WriteTransaction(t => {
            var temp = t.Run("match(n:"+label+")-[r]->(i) return distinct type(r) as type").ToList();
            return temp;
        });
        foreach (var tem in result)
        {
            string str;
            str = JsonConvert.DeserializeObject<string>(JsonConvert.SerializeObject(tem[0]));
            list.Add(str);
        }
    }
    return list;
}
```

```
public void InitiateDatabase()
{
    using (var session = _driver.Session())
    {
        session.WriteTransaction(tx =>
        {
            tx.Run("Load csv with headers from " + @"file:///producers.csv" + "as n "+
                    "create(s: Producers)" +
                    "set s = n;");
            tx.Run("Load csv with headers from " + @"file:///PrimaryConsumers.csv" + "as n " +
                     "create(s: Producers)" +
                     "set s = n;");
            tx.Run("Load csv with headers from " + @"file:///SecondaryConsumers.csv" + "as n " +
                    "create(s: Producers)" +
                    "set s = n;");
            tx.Run("Load csv with headers from " + @"file:///TertiaryConsumers.csv" + "as n " +
                    "create(s: Producers)" +
                    "set s = n;");
            tx.Run("Match (p:Producers),(pr:PrimaryConsumers)" +
                "Create(p) -[:Provide]->(pr)" +
                "Create(pr) -[:Consume]->(p)");
            tx.Run("Match (p:PrimaryConsumers),(s:SecondaryConsumers)" +
                "Create(p) -[:Provide]->(s)" +
```

Load dữ liệu từ các tệp csv và tạo csdl trên neo4j

Tài liệu tham khảo:

Neo4j - Getting Started, *Why Graph Databases?* , *What is a Graph Database?*, *Concepts: RDBMS to Graph*, *Concepts: NoSQL to Graph* *(2018)*

Link: https://neo4j.com/developer/get-started/