**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO CUỐI KỲ MÔN HỌC BIG DATA ANALYSIS**

**ĐỀ TÀI: Cài đặt và truy vấn CSDL NoSQL Neo4j**

**GVHD: ThS. LÊ THỊ MINH CHÂU**

**Lớp: Thứ 2 (tiết 3 - 6)**

**Sinh viên thực hiện: (Nhóm 15)**

**Lê Lương Trường An (MSSV: 21133001)**

**Nguyễn Đức Duy (MSSV: 21133020)**

**Trương Võ Toại (MSSV: 19133057)**

**Nguyễn Quốc Thắng (MSSV: 20133120)**

**Thành phố Hồ Chí Minh, tháng 12 năm 2023**

|  |  |
| --- | --- |
| **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**  **THÀNH PHỐ HỒ CHÍ MINH**  **KHOA CÔNG NGHỆ THÔNG TIN**  **BỘ MÔN BIG DATA ANALYSIS** | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập - Tự do - Hạnh phúc**  *Thành phố Hồ Chí Minh, tháng 12 năm 2023* |
|  |  |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

1. **Môn học*:*** *BIG DATA ANALYSIS* *(Thứ 2, tiết 3 - 6)*
2. **Giảng viên hướng dẫn:** *ThS. Lê Thị Minh Châu*
3. **Đề tài:** *Cài đặt và truy vấn CSDL NoSQL Neo4j*
4. **Danh sách nhóm 15:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **HỌ VÀ TÊN SINH VIÊN** | **Mã số sinh viên** | **Tỉ lệ tham gia %** | **Kí tên** |
| 1 | Lê Lương Trường An | 21133001 | 100% |  |
| 2 | Nguyễn Đức Duy | 21133020 | 100% |  |
| 3 | Trương Võ Toại | 19133057 | 100% |  |
| 4 | Nguyễn Quốc Thắng | 20133120 | 100% |  |

**Nhận xét:**

**Điểm:**

**Thành phố Hồ Chí Minh, tháng 12 năm 2023**

*Giáo viên hướng dẫn ký tên*

*(Ký & ghi rõ họ tên)*

**BẢNG PHÂN CÔNG CÔNG VIỆC**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | MSSV | Họ và tên | Phân công nhiệm vụ |
| 1 | 21133001 | Lê Lương Trường An | Làm file word, tìm nội dung  Demo Xóa nút và quan hệ  (Chương 2 – mục 2.2.3) |
| 2 | 21133020 | Nguyễn Đức Duy(NT) | Tìm tài liệu, chỉnh sửa nội dung  Demo thao tác tạo hoặc sửa nút và quan hệ (Chương 2 – mục 2.2.1, 2.2.2) |
| 3 | 19133057 | Trương Võ Toại | Tìm nội dung  Demo Truy vấn và lọc trên dữ liệu có sẵn  (Chương 2 – mục 2.2.5) |
| 4 | 20133120 | Nguyễn Quốc Thắng | Tìm nội dung  Demo Lọc các nút  (Chương 2 – mục 2.2.4) |

**LỜI CẢM ƠN**

Lời đầu tiên, nhóm muốn gửi lời cảm ơn chân thành và sâu sắc nhất đến cô Lê Thị Minh Châu - giảng viên bộ môn Big Data Analysis.

Thứ hai, nhóm cũng muốn bày tỏ lòng biết ơn chân thành đến các thành viên và các nhóm trong lớp học đã gửi câu hỏi và đóng góp ý kiến đến nhóm. Nhờ những ý kiến này, nhóm đã hoàn thiện và khắc phục những thiếu sót.

Nhóm cũng muốn gửi lời cảm ơn đặc biệt đến cô vì sự quan tâm và sự giúp đỡ tận tâm của cô trong suốt quá trình giảng dạy. Nhóm trân trọng cô đã luôn trả lời những thắc mắc của nhóm và đưa ra nhận xét, góp ý để nhóm cải thiện chất lượng công việc của nhóm.

Vì khả năng của nhóm còn hạn chế, trong quá trình thực hiện báo cáo không tránh khỏi sai sót. Nhóm mong nhận được ý kiến đóng góp từ cô để nhóm có thể cải thiện hơn trong tương lai.

Nhóm thực hiện bài báo cáo xin chân thành cảm ơn cô.

**Nhóm 15**

Lê Lương Trường An

Nguyễn Đức Duy

Trương Võ Toại

Nguyễn Quốc Thắng

MỤC LỤC

[PHẦN 1: MỞ ĐẦU 1](#_Toc153575361)

[1. Mục đích của đề tài 1](#_Toc153575362)

[2. Đối tượng và phạm vi nghiên cứu 1](#_Toc153575363)

[3. Kết quả dự kiến đạt được 1](#_Toc153575364)

[PHẦN 2: NỘI DUNG 2](#_Toc153575365)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 2](#_Toc153575366)

[1.1. Sơ lược về NoSQL 2](#_Toc153575367)

[1.1.1. Giới thiệu chung 2](#_Toc153575368)

[1.2. Các loại cơ sở dữ liệu NoSQL 2](#_Toc153575369)

[1.2.1. Key-value stores 2](#_Toc153575370)

[1.2.2. Column-oriented database 2](#_Toc153575371)

[1.2.3. Graph databases 3](#_Toc153575372)

[1.2.4. Document Oriented databases 3](#_Toc153575373)

[1.3. Sơ lược về Neo4j 3](#_Toc153575374)

[1.3.1. Giới thiệu về Neo4j 3](#_Toc153575375)

[1.3.2. Các thành phần của Nền tảng đồ thị Neo4j 4](#_Toc153575376)

[1.3.3. Các thành phần của cơ sở dữ liệu Neo4j 7](#_Toc153575377)

[1.3.4. Giới thiệu về Cypher 8](#_Toc153575378)

[CHƯƠNG 2: CÀI ĐẶT VÀ THIẾT KẾ 10](#_Toc153575379)

[2.1. Cài đặt Neo4j 10](#_Toc153575380)

[2.2. Một số thao tác trong Neo4j 14](#_Toc153575381)

[2.2.1.Tạo nút và quan hệ 14](#_Toc153575382)

[2.2.2. Sửa nút và quan hệ 15](#_Toc153575383)

[2.2.3. Xóa nút và quan hệ 16](#_Toc153575384)

[2.2.4. Lọc các nút. 19](#_Toc153575385)

[2.2.5. Truy vấn và lọc trên dữ liệu có sẵn 24](#_Toc153575386)

[PHẦN 3: KẾT LUẬN 31](#_Toc153575387)

[1. Kết quả đạt được 31](#_Toc153575388)

[2. Về demo 31](#_Toc153575389)

[3. Hạn chế 31](#_Toc153575390)

[4. Hướng phát triển 31](#_Toc153575391)

[TÀI LIỆU THAM KHẢO 32](#_Toc153575392)

# PHẦN 1: MỞ ĐẦU

## 1. Mục đích của đề tài

“Cơ sở dữ liệu” thuật ngữ đã trở nên rất quen thuộc với các nhà phát triển ứng dụng có rất nhiều dạng cơ sở dữ liệu như: Cơ sở dữ liệu quan hệ, cơ sở dữ liệu hướng đối tượng, cơ sở dữ liệu không gian, cơ sở dữ liệu phi cấu trúc,… một dạng cơ sở dữ liệu thường được sử dụng trong việc môt các các mạng thông tin như mạng xã hội, mạng cảm biến,… các dạng mà ở đó thông tin được mô tả như một thực thể trên mộ đồ thị có hướng đó là cơ sơ dữ liệu đồ thị Neo4j.

## 2. Đối tượng và phạm vi nghiên cứu

* Đối tượng nghiên cứu: Đối tượng nghiên cứu của đồ án là tìm hiểu, phân tích và sử dụng NoSQL Neo4j.
* Phạm vi nghiên cứu: Đồ án chỉ nghiên cứu trong phạm vi ứng dụng của Neo4j platform

## 3. Kết quả dự kiến đạt được

* Tìm hiểu và phân tích sử dụng các thành phần của Neo4j

# PHẦN 2: NỘI DUNG

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

## 1.1. Sơ lược về NoSQL

### 1.1.1. Giới thiệu chung

Cơ sở dữ liệu NoSQL là Cơ sở dữ liệu được xây dựng dành riêng cho mô hình dữ liệu và có sơ đồ linh hoạt để xây dựng các ứng dụng hiện đại. Cơ sở dữ liệu NoSQL được công nhận rộng rãi vì khả năng dễ phát triển, chức năng cũng như hiêu năng ở quy mô lớn. Trang này có các tài nguyên giúp bạn hiểu thêm về cơ sở dữ liệu NoSQL và bắt đầu sử dụng.

Cơ sở dữ liệu NoSQL sử dụng nhiều mô hình dữ liệu để truy cập và quản lý dữ liệu. Các loại cơ sở dữ liệu này được tối ưu hóa dành riêng cho các ứng dụng yêu cầu mô hình dữ liệu linh hoạt có lượng dữ liệu lớn và độ trễ thấp, có thể đạt được bằng cách giảm bớt một số hạn chế về tính nhất quán của dữ liệu của các cơ sở dữ liệu khác.

## 1.2. Các loại cơ sở dữ liệu NoSQL

### 1.2.1. Key-value stores

* Lưu trữ kiểu key-value là kiểu lưu trữ dữ liệu NoSQL đơn giản nhất sử dụng từ một API. Chúng ta có thể nhận được giá trị cho khóa, đặt một giá trị cho một khóa, hoặc xóa một khóa từ dữ liệu.
* Một vài cơ sở dữ liệu key-value phổ biến là Riak, Redis, memcached, Berkeley DB, HamsterDB, Amazon DynamoDB, Project Voldemort và Couchbase.

### 1.2.2. Column-oriented database

Cơ sở dữ liệu column-family lưu trữ dữ liệu trong nhiều cột trong mỗi dòng với key cho từng dòng. Column families là một nhóm các dữ liệu liên quan được truy cập cùng với nhau. Ví dụ, với khách hàng, chúng ta thường xuyên sử dụng thông tin cá nhân trong cùng một lúc chứ không phải hóa đơn của họ. Cassandra là một trong số cơ sở dữ liệu column-family phổ biến. Ngoài ra còn có một số cơ sở dữ liệu khác như HBase, Hypertable và Amazon DynamoDB. Cassandra có thể được miêu tả nhanh và khả năng mở rộng dễ dàng với các thao tác viết thông qua các cụm. Các cụm không có node master, vì thế bất kỳ việc đọc và ghi nào đểu có thể được xử lý bởi bất kỳ node nào trong cụm.

### 1.2.3. Graph databases

Kiểu đồ thị này cho phép bạn lưu trữ các thực thể và quan hệ giữa các thực thể. Các đối tượng này còn được gọi là các nút, trong đó có các thuộc tính. Mỗi nút là một thể hiện của một đối tượng trong ứng dụng. Quan hệ được gọi là các cạnh, có thể có các thuộc tính. Cạnh có ý nghĩa định hướng; các nút được tổ chức bởi các mối quan hệ. Các tổ chức của đồ thị cho phép các dữ liệu được lưu trữ một lần và được giải thích theo nhiều cách khác nhau dựa trên các mối quan hệ.

Một số cơ sở dữ liệu tài liệu phổ biến mà chúng ta hay gặp là Neo4J, Infinite Graph, OrientDB, hoặc FlockDB

### 1.2.4. Document Oriented databases

NoSQL Document Database lưu trữ và truy xuất dữ liệu dưới dạng một cặp giá trị khóa (key value pair) nhưng phần giá trị được lưu trữ dưới dạng tài liệu. Tài liệu được lưu trữ ở định dạng JSON hoặc XML. Giá trị được hiểu bởi Online Transaction Processing DB và có thể được truy vấn bởi một số cơ sở dữ liệu tài liệu phổ biến mà chúng ta hay gặp là MongoDB, CouchDB, Terastore, OrientDB, RavenDB.

## 1.3. Sơ lược về Neo4j

### 1.3.1. Giới thiệu về Neo4j

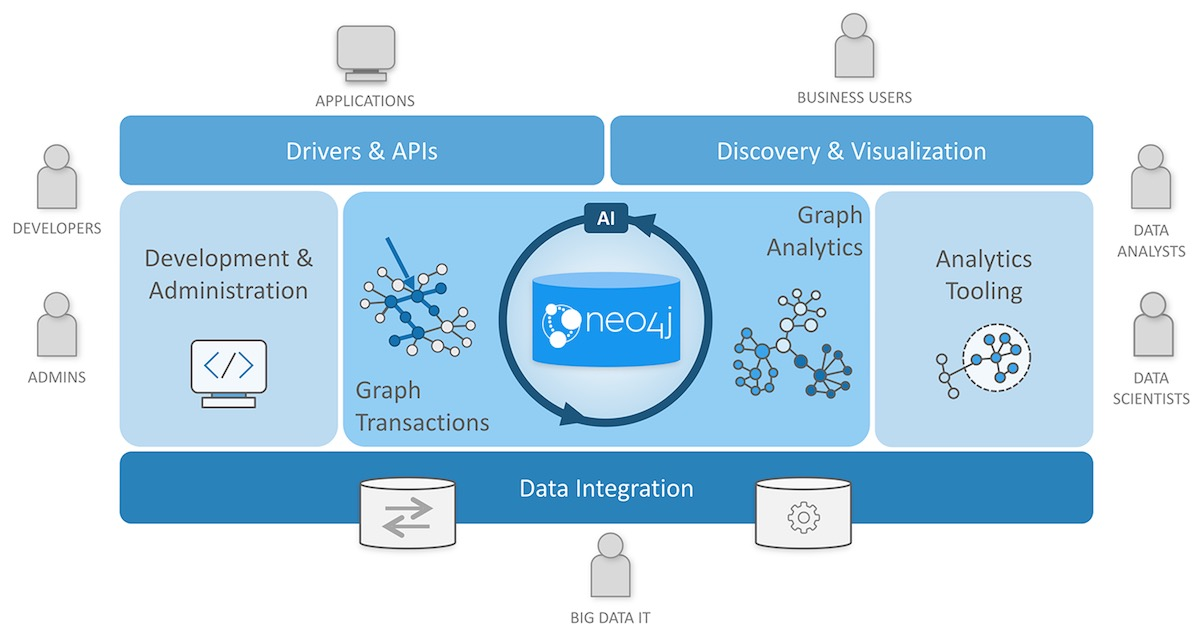
Neo4j là cơ sở dữ liệu đồ thị, mã nguồn mở NoSQL, được công bố từ năm 2007. Neo4j được cung cấp như một dịch vụ được quản lý thông qua AuraDB. Tuy nhiên, bạn cũng có thể tự mình chạy Neo4j với Phiên bản Community hoặc Phiên bản Enterprise. Phiên bản Enterprise bao gồm tất cả những gì Community Edition phải cung cấp, cùng với các yêu cầu bổ sung dành cho doanh nghiệp như khả năng sao lưu, phân cụm và chuyển đổi dự phòng. Neo4j được viết bằng Java và Scala, và mã nguồn có sẵn trên GitHub.

Neo4j là một cơ sở dữ liệu đồ thị gốc, có nghĩa là nó triển khai một mô hình đồ thị thực sự cho đến tận cấp độ lưu trữ. Dữ liệu không được lưu trữ dưới dạng "trừu tượng hóa biểu đồ" trên công nghệ khác, nó được lưu trữ giống như bạn viết bảng trắng. Điều này rất quan trọng vì đó là lý do tại sao Neo4j hoạt động tốt hơn các biểu đồ khác và luôn linh hoạt. Ngoài biểu đồ, Neo4j cung cấp cả cơ sở dữ liệu; Các giao dịch ACID, hỗ trợ cụm và chuyển đổi dự phòng.

Những đặc điểm quan trọng của Neo4j:

* Cypher, một ngôn ngữ truy vấn khai báo tương tự như SQL, nhưng được tối ưu hóa cho đồ thị
* Thời gian liên tục di chuyển trong các biểu đồ lớn cho cả chiều sâu và chiều rộng do biểu diễn hiệu quả các nút và mối quan hệ. Cho phép mở rộng quy mô lên đến hàng tỷ nút trên phần cứng vừa phải.
* Lược đồ biểu đồ thuộc tính linh hoạt có thể thích ứng theo thời gian, giúp bạn có thể hiện thực hóa, thêm các mối quan hệ mới sau này để tắt và tăng tốc dữ liệu miền khi nhu cầu kinh doanh thay đổi
* Driver cho các ngôn ngữ lập trình phổ biến, bao gồm Java, JavaScript, .NET, Python, v.v.

### 1.3.2. Các thành phần của Nền tảng đồ thị Neo4j



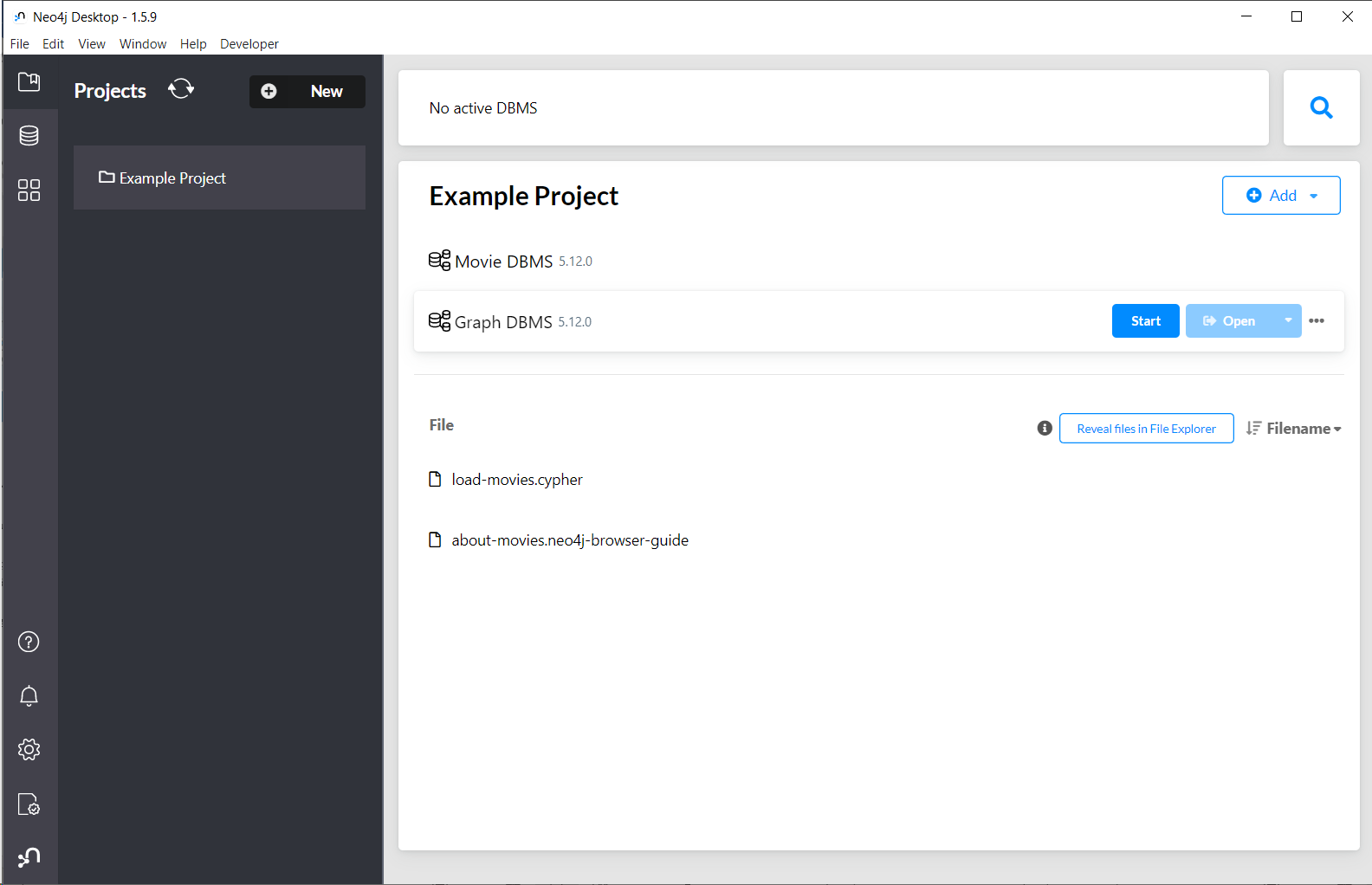
#### 1.3.2.1. Neo4j Graph Database

Cơ sở dữ liệu đồ thị cốt lõi được xây dựng để lưu trữ và truy xuất dữ liệu được kết nối.

Có 2 phiên bản: Phiên bản Cộng đồng và Phiên bản Doanh nghiệp. Mọi thứ trong nền tảng đều tương tác với dữ liệu được lưu trữ trong cơ sở dữ liệu.

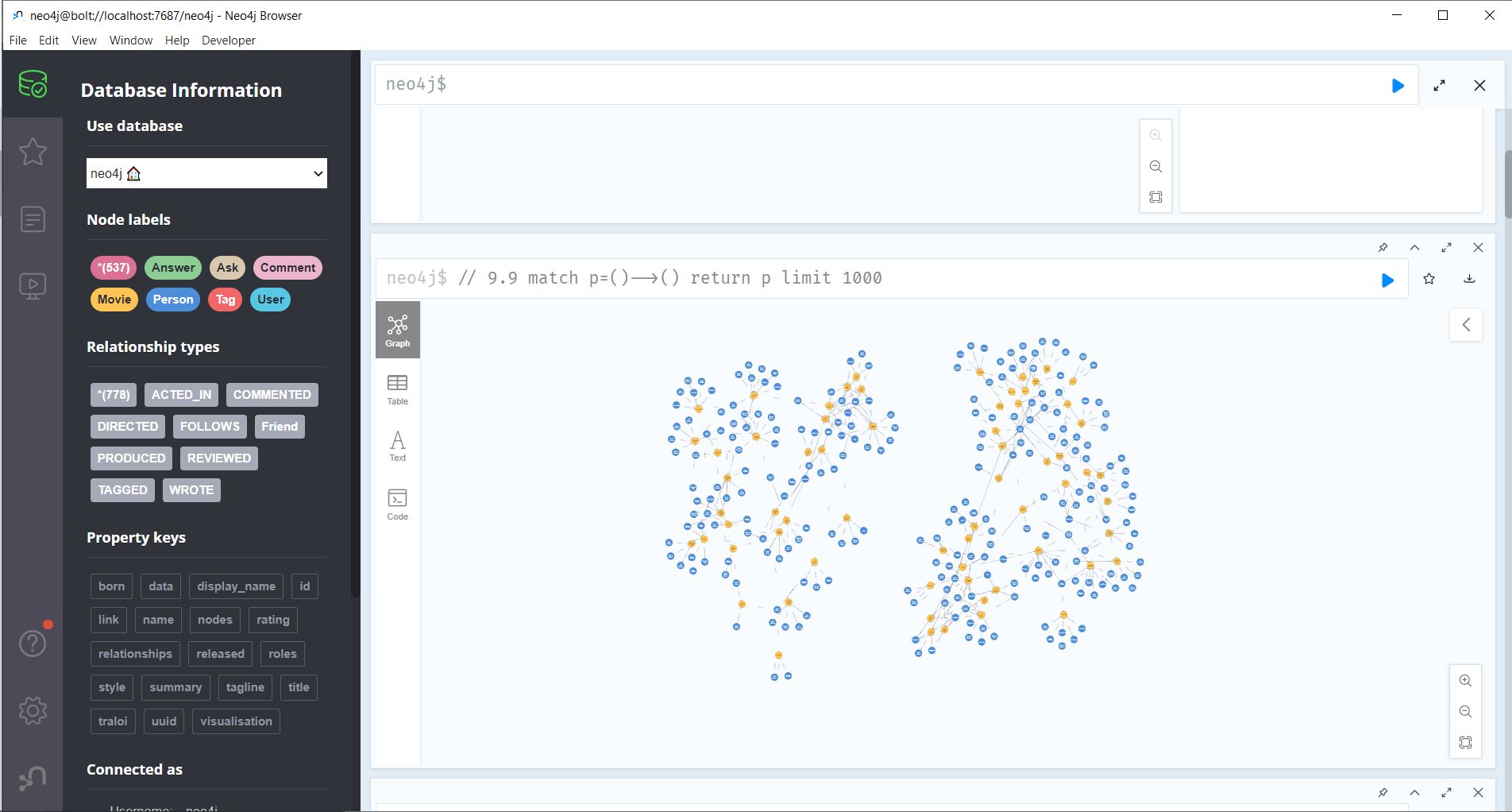
#### 1.3.2.2. Neo4j Desktop

Ứng dụng để quản lý các phiên bản cục bộ của Neo4j. Tải xuống miễn phí bao gồm giấy phép Neo4j Enterprise Edition.



#### 1.3.2.3. Neo4j Browser

Giao diện trình duyệt trực tuyến để truy vấn và xem dữ liệu trong cơ sở dữ liệu. Khả năng trực quan hóa cơ bản sử dụng ngôn ngữ truy vấn Cypher.



#### 1.3.2.4. Neo4j Bloom

Công cụ trực quan hóa cho người dùng doanh nghiệp không yêu cầu bất kỳ mã hoặc kỹ năng lập trình nào để xem và phân tích dữ liệu.



#### 1.3.2.5. Neo4j AuraDB

Cung cấp cơ sở dữ liệu dưới dạng dịch vụ được quản lý bởi Neo4j cho cơ sở dữ liệu đồ thị trên đám mây.

#### 1.3.2.6. Graph Data Science

Thư viện được hỗ trợ chính thức để thực thi các thuật toán đồ thị với Neo4j và được tối ưu hóa cho khối lượng công việc và đường ống của doanh nghiệp.

#### 1.3.2.7. Các công cụ tích hợp trong Neo4j

Neo4j cũng có nhiều thư viện mở rộng và công cụ dành cho nhà phát triển có thể được thêm vào các sản phẩm hiện có để nâng cao chức năng.

Tích hợp Neo4j Labs:

* APOC - thư viện tiện ích tiêu chuẩn của các thủ tục và chức năng cho Neo4j.
* GraphQL và GRANDstack - tích hợp với các công nghệ dữ liệu phổ biến khác hoặc các giải pháp toàn ngăn xếp.
* Công cụ ETL - di chuyển dữ liệu từ cơ sở dữ liệu quan hệ sang Neo4j bằng ứng dụng và giao diện người dùng đơn giản

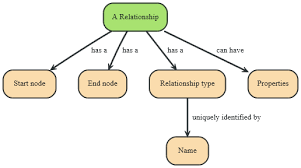
### 1.3.3. Các thành phần của cơ sở dữ liệu Neo4j

#### 1.3.3.1. Node

Trong Neo4j các đơn vị cơ bản để hình thành một đồ thị là các nodes và các relationship. Cả nodes và relationship đều có thể chứa các thuộc tính.

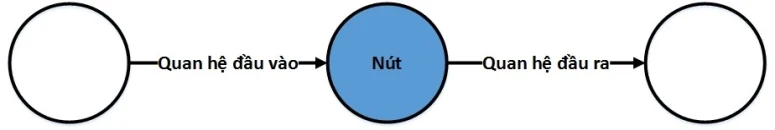
Nodes thường được sử dụng để biểu diễn các thực thể (entities). Đồ thị đơn giản nhất là đồ thì mà trong đó chỉ có duy nhất một node.

#### 1.3.3.2. Relationships

 Biểu diễn mối quan hệ, hay liên kết giữa các node với nhau. Ngoài ra relationship còn có thể cấu trúc phân chia các nodes thành những cấu trúc khác nhau, biến đồ thị thành các dạng cấu trúc giống như là list, tree, map, hoặc có thể là thực thể phức hợp (compound entity). Thực thể phức hợp là thực thể có nhiều liên kết phức tạp liên kết với nhau.

Trong Neo4j, mỗi Relationship là một cạnh có hướng nối Node nguồn và Node

đích (một Node có thể có cạnh nối với chính nó).



#### 1.3.3.3. Relationship Types

Mỗi Relationship chỉ có duy nhất một Relationship Types. Relationship Type giúp ta biết được mối quan hệ giữa các Node là gì. Việc đặt type cho Relationship giúp chúng ta dễ hình dung được Node nào là Node nguồn và Node nào là Node đích.

#### 1.3.3.4. Properties

Properties là một cặp key-value thể hiện các thuộc tính của Node hoặc Relationship. Các value có thể có các kiểu dữ liệu như number, string, boolean hoặc list. “null” không phải là giá trị thuộc tính hợp lệ. Null có thể được mô hình hóa bởi một … (the absence of key).

#### 1.3.3.5. Traversals

Traversal là cách mà chúng ta truy vấn dữ liệu để trả lời cho một câu hỏi. Travesaling một đồ thị nghĩa là chúng ta bắt đầu từ một Node và lần theo các Relationship theo các quy tác nào đó. Hầu như chúng ta chỉ cần lần theo Đồ thị con của Đồ thị (dựa vào các Label, các Relationship Type).

#### 1.3.3.6. Paths

Kết quả của traversal có thể trả về là một đường đi từ Node này đến Node kia.

### 1.3.4. Giới thiệu về Cypher

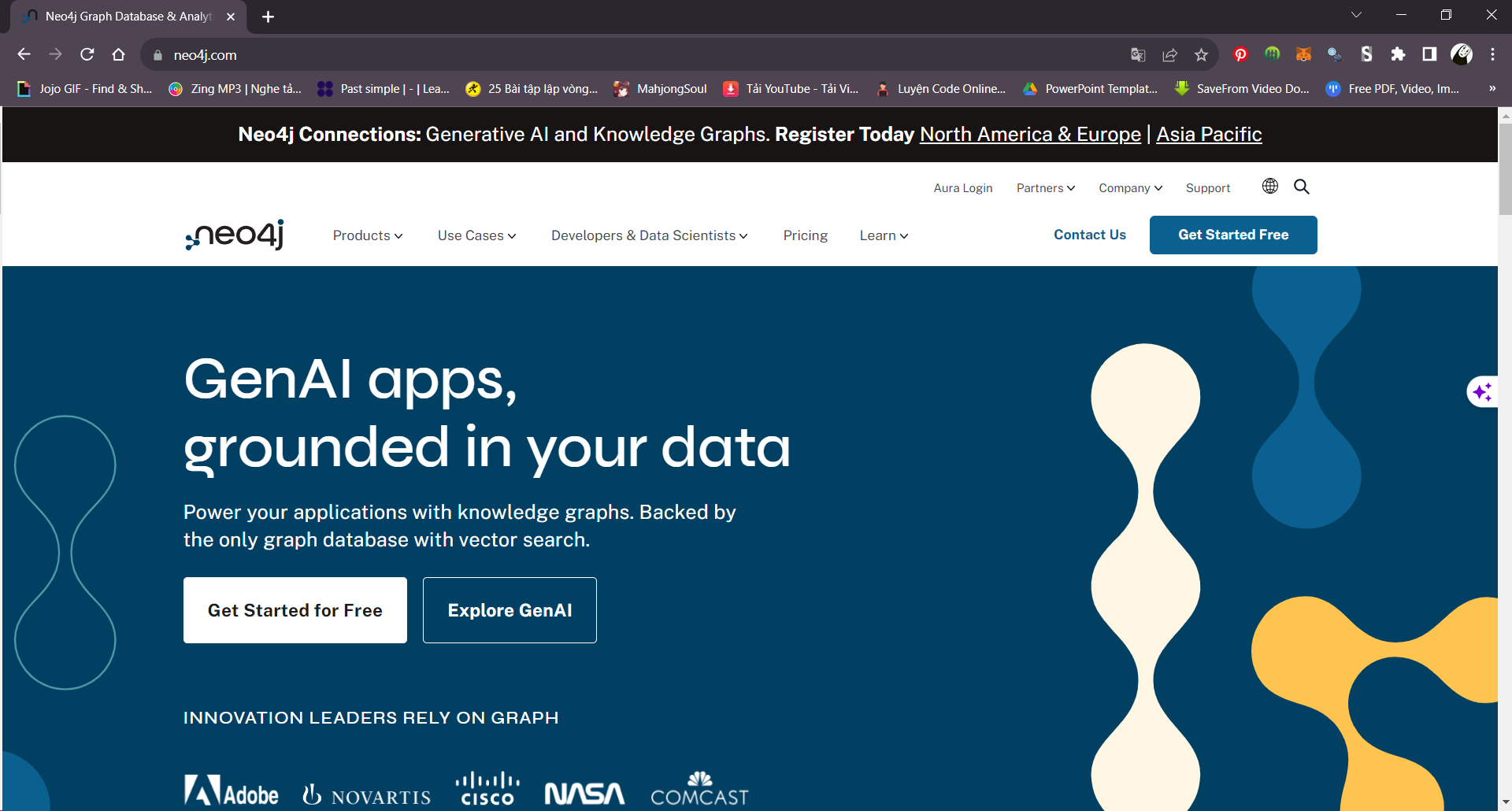
Cypher là ngôn ngữ truy vấn trong Neo4j. Các câu query là tập hợp các mệnh đề được liên kết với nhau.

* MATCH: Tìm kiếm theo pattern. OPTIONAL MATCH tương tự MATCH nhưng sẽ trả về kết quả Null nếu có missing trong pattern.
* MERGE: Đảm bảo pattern luôn tồn tại trong đồ thị, nếu không tồn tại, MERGE sẽ giúp tạo pattern đó.
* WITH: Thực hiện các thao tác với các output trước khi sang các mệnh đề khác.
* WHERE: Thêm các ràng buộc cho pattern.
* RETURN: Định nghĩa kết quả trả về.
* CREATE: Tạo Node, Relationship hoặc một Path.
* DELETE: Xóa Node, Relationship.
* SET: Dùng để cập nhật Labels, Properties.
* LIMIT, ORDER BY: tương tự như SQL

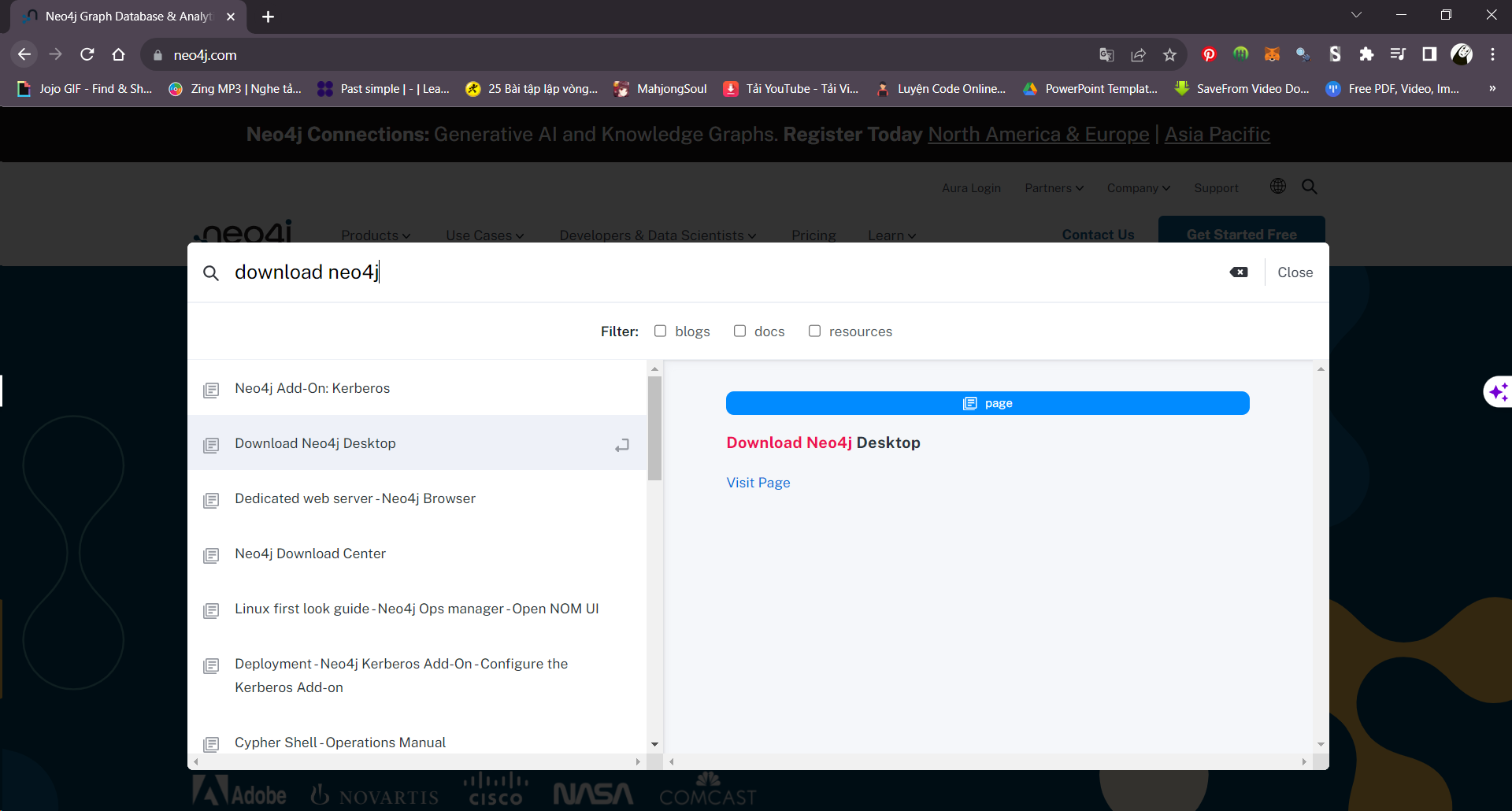
# CHƯƠNG 2: CÀI ĐẶT VÀ THIẾT KẾ

## 2.1. Cài đặt Neo4j

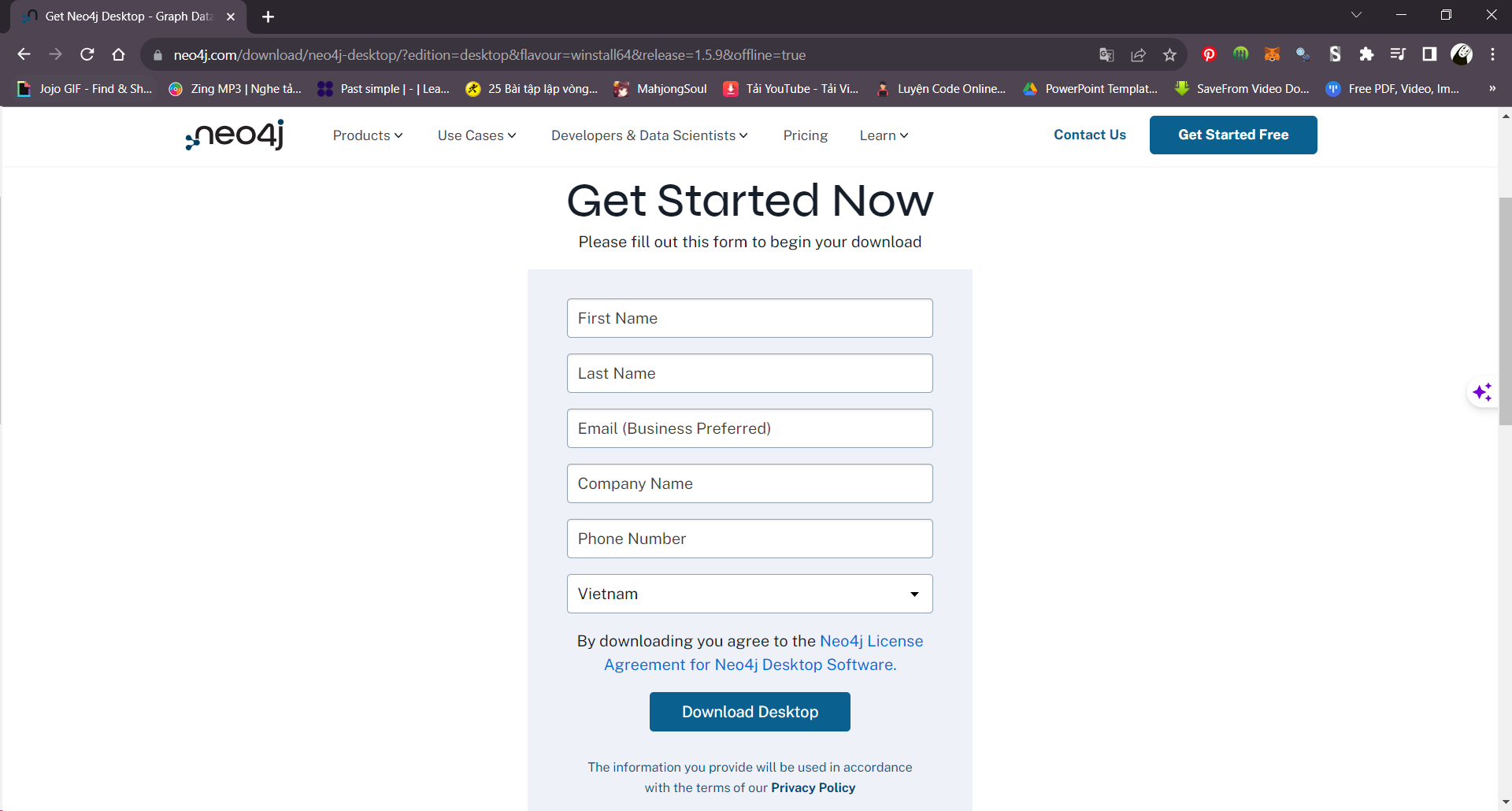
Đầu tiên, truy cập vào website [neo4j.com](https://neo4j.com/)



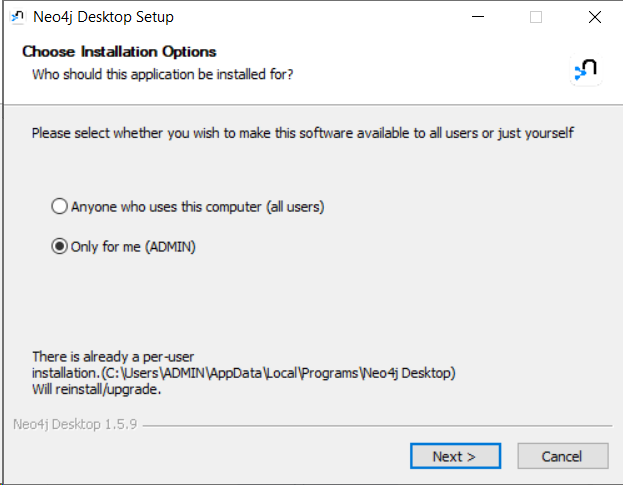
Tiếp theo, trên thanh tìm kiếm ta ghi: Download Neo4j

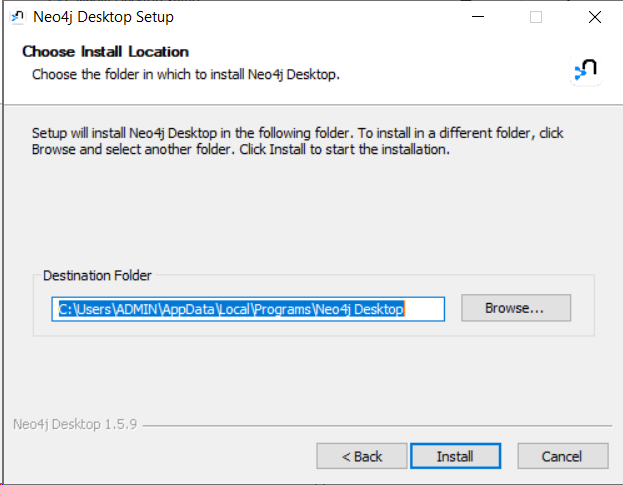


Sau đó, sẽ hiện ra cửa sổ mới chúng ta nhập thông tin cá nhân mà website yêu cầu và ấn Download Desktop để tải về máy.

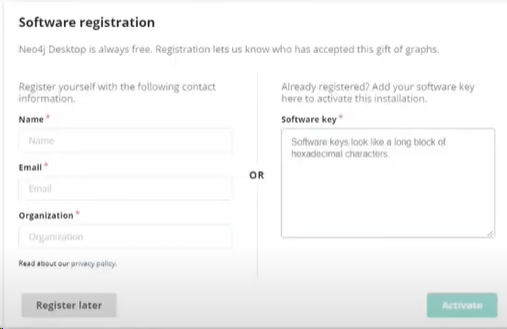


Có thể giữ các đề nghị tải xuống của Neo4j hoặc có thể thay đổi theo nhu cầu sử dụng của bản thân.





Sau khi cài đặt xong, khi mở Neo4j lên sẽ nhập lại một số thông tin cá nhân lúc nãy bên website yêu cầu chúng ta nhập.



## 2.2. Một số thao tác trong Neo4j

### 2.2.1.Tạo nút và quan hệ

Thực hiện các công việc sau:

* + Tạo Nút Người (Person Node): Tạo một nút trong đồ thị với nhãn Person và gán cho nó một thuộc tính name có giá trị "Duc Duy". Nút này đại diện cho một người có tên là Duc Duy.
  + Tạo Nút Phim (Movie Node): Tạo một nút khác trong đồ thị với nhãn Movie và gán cho nó một thuộc tính title có giá trị "Film Red". Nút này đại diện cho một bộ phim có tên là Film Red.
  + Tạo Quan Hệ Diễn Xuất (ACTED\_IN Relationship): Tạo một quan hệ từ nút người (Duc Duy) đến nút phim (Film Red) với nhãn ACTED\_IN. Quan hệ này còn chứa một thuộc tính role có giá trị "Lu", cho thấy Duc Duy đã diễn xuất trong vai Lu trong phim Film Red.
  + Trả Về Kết Quả: Sau khi tạo nút và quan hệ, lệnh RETURN Duy, Red sẽ trả về thông tin của nút người (Duc Duy) và nút phim (Film Red). Điều này giúp xác nhận rằng các nút và quan hệ đã được tạo thành công và cung cấp thông tin về chúng.

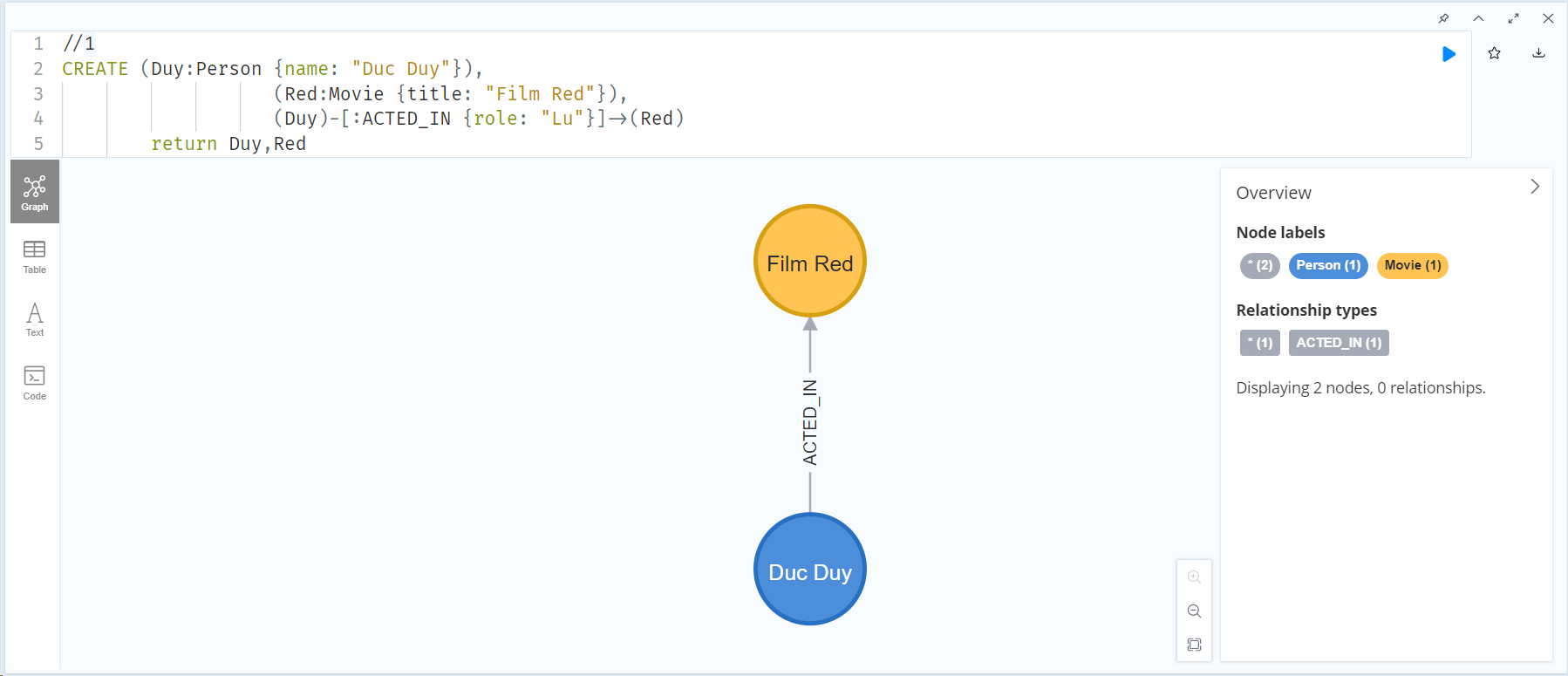
Truy vấn :

**CREATE (Duy:Person {name: "Duc Duy"}),**

**(Red:Movie {title: "Film Red"}),**

**(Duy)-[:ACTED\_IN {role: "Lu"}]->(Red)**

**return Duy,Red**



### 2.2.2. Sửa nút và quan hệ

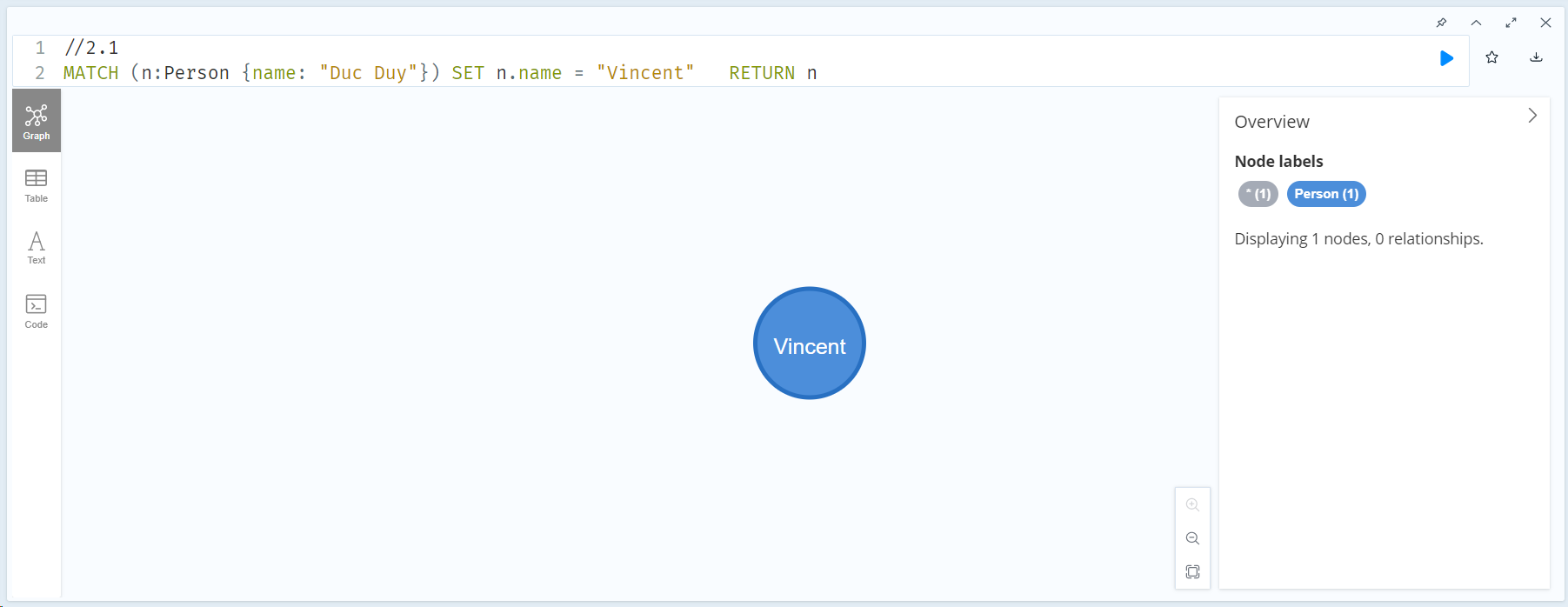
Để thay đổi tên nút ta dùng cách sau:

* + Tìm Kiếm Nút (Matching the Node): Lệnh MATCH (n:Person {name: "Duc Duy"}) tìm kiếm trong cơ sở dữ liệu đồ thị cho tất cả các nút có nhãn Person và có thuộc tính name giá trị là "Duc Duy".
  + Cập Nhật Thuộc Tính (Updating the Property): SET n.name = "Vincent" sẽ cập nhật thuộc tính name của nút/nút được tìm thấy từ "Duc Duy" thành "Vincent".
  + Trả Về Kết Quả (Returning the Node): RETURN n sẽ trả về nút/nút sau khi đã được cập nhật.

Truy vấn:

**MATCH (n:Person {name: "Duc Duy"}) SET n.name = "Vincent”**

**RETURN n**



Để thay đổi quan hệ giữ hai nút ta dùng cách sau:

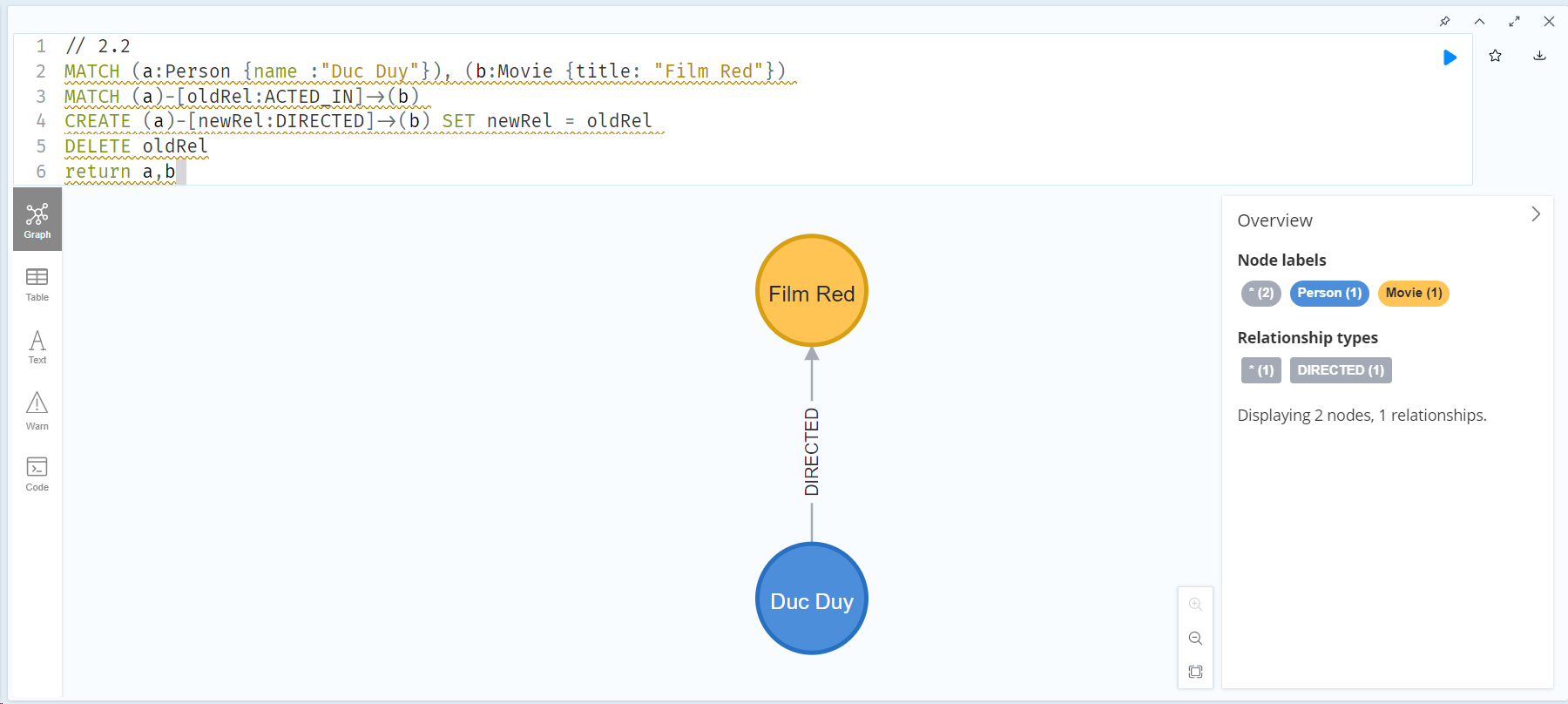
**MATCH (a:Person {name :"Duc Duy"}), (b:Movie {title: "Film Red"}) MATCH (a)-[oldRel:ACTED\_IN]->(b)**

**CREATE (a)-[newRel:DIRECTED]->(b) SET newRel = oldRel**

**DELETE oldRel**

Trong đoạn truy vấn trên:

* + Đầu tiên, tìm hai nút a và b mà bạn muốn thay đổi quan hệ.
  + Tìm quan hệ hiện tại OLD\_RELATIONSHIP giữa a và b.
  + Tạo một quan hệ mới NEW\_RELATIONSHIP giữa a và b, và sao chép tất cả thuộc tính từ quan hệ cũ sang quan hệ mới (nếu có).
  + Xóa quan hệ cũ.



### 2.2.3. Xóa nút và quan hệ

#### 2.2.3.1. Xóa nút đơn

Để xóa các nút và mọi mối quan hệ kết nối chúng, hãy sử dụng DETACH DELETE mệnh đề.

**Truy vấn:**

**MATCH (n:Person {name:"Vincent"}) DETACH DELETE n**

Person Thao tác này sẽ xóa nút Vincent và tất cả các mối quan hệ được kết nối với nó.

A black rectangular object with a black stripe

Description automatically generated

#### 2.2.3.2. Xóa quan hệ

Để xóa một quan hệ trong Neo4j : sử dụng ngôn ngữ truy vấn Cypher, bạn cần xác định quan hệ cần xóa thông qua một truy vấn MATCH và sau đó sử dụng lệnh DELETE để xóa nó.

* + Xác Định Quan Hệ: Sử dụng MATCH để tìm quan hệ cần xóa. Bạn cần xác định rõ các nút và loại quan hệ.
  + Xóa Quan Hệ: Sau khi tìm được quan hệ, sử dụng DELETE để xóa nó.

A screenshot of a computer

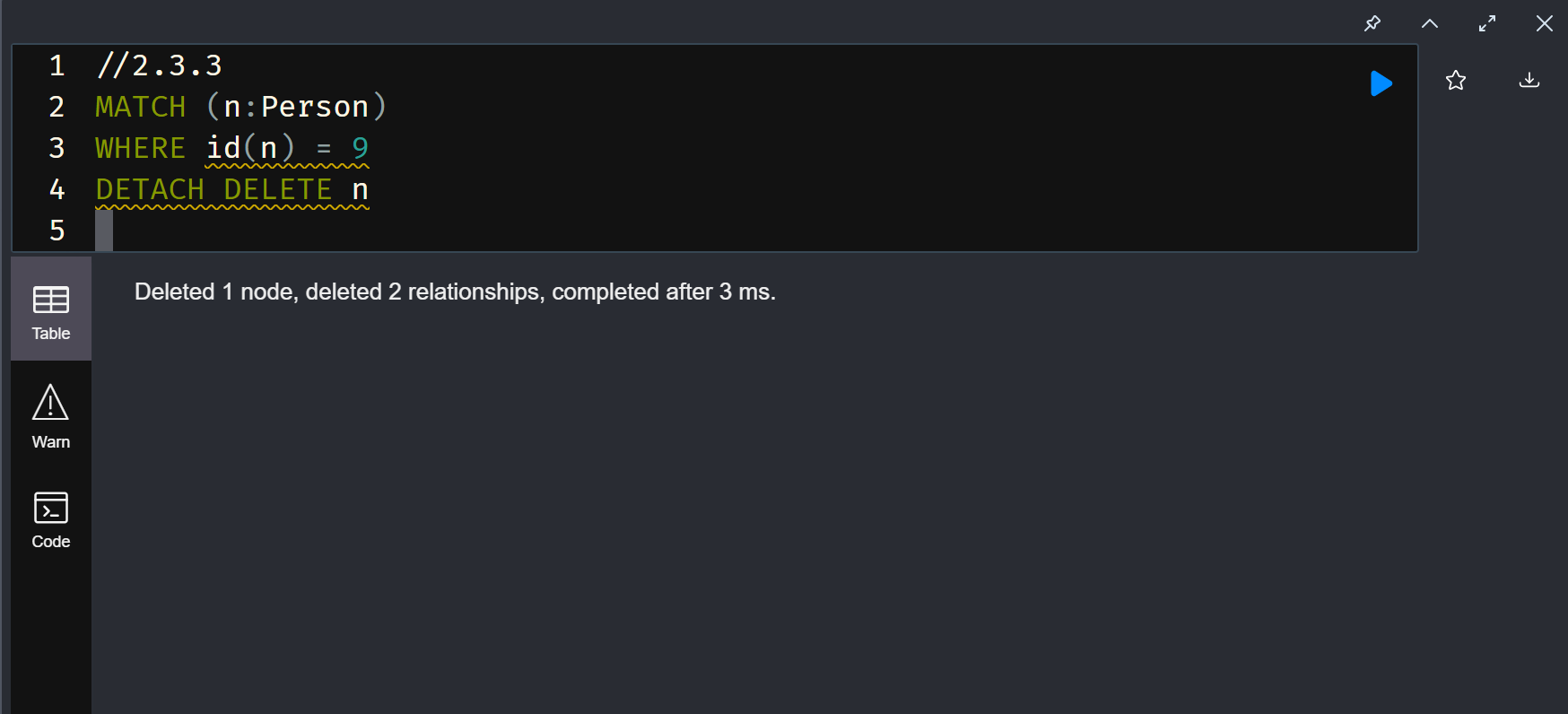
Description automatically generated

#### 2.2.3.3 Xóa nút theo điều kiện

Xóa Nút Dựa Trên ID: Sử dụng hàm ID() để tìm nút với ID cụ thể và sau đó xóa nó. Lưu ý rằng bạn cần đảm bảo rằng không có mối quan hệ nào liên kết với nút này trước khi xóa, hoặc sử dụng DETACH DELETE để tự động xóa cả nút lẫn các mối quan hệ liên quan.

**Truy vấn:**

**MATCH (n) WHERE ID(n) = 123 DETACH DELETE n**



#### 2.2.3.4 Xóa mối quan hệ động

Để xóa mối quan hệ động, bạn sẽ sử dụng một truy vấn Cypher mà ở đó loại mối quan hệ được tham chiếu thông qua một biến. Dưới đây là một số bước cơ bản:

* Xác Định Loại Mối Quan Hệ: Đầu tiên, bạn cần xác định loại mối quan hệ cần xóa. Điều này có thể dựa trên một điều kiện hoặc một giá trị động.
* Sử Dụng TYPE() để Lấy Loại Mối Quan Hệ: Sử dụng hàm TYPE() trong truy vấn Cypher để lấy loại mối quan hệ của mỗi mối quan hệ.
* So Sánh và Xóa: So sánh loại mối quan hệ thu được với điều kiện của bạn và thực hiện thao tác xóa nếu nó phù hợp.

Ví dụ:

Giả sử bạn muốn xóa tất cả mối quan hệ 'FRIEND' của một nút cụ thể:

**Truy vấn:**

**MATCH (p:Person {name: 'John'})-[r]->()**

**WHERE TYPE(r) = 'FRIEND'**

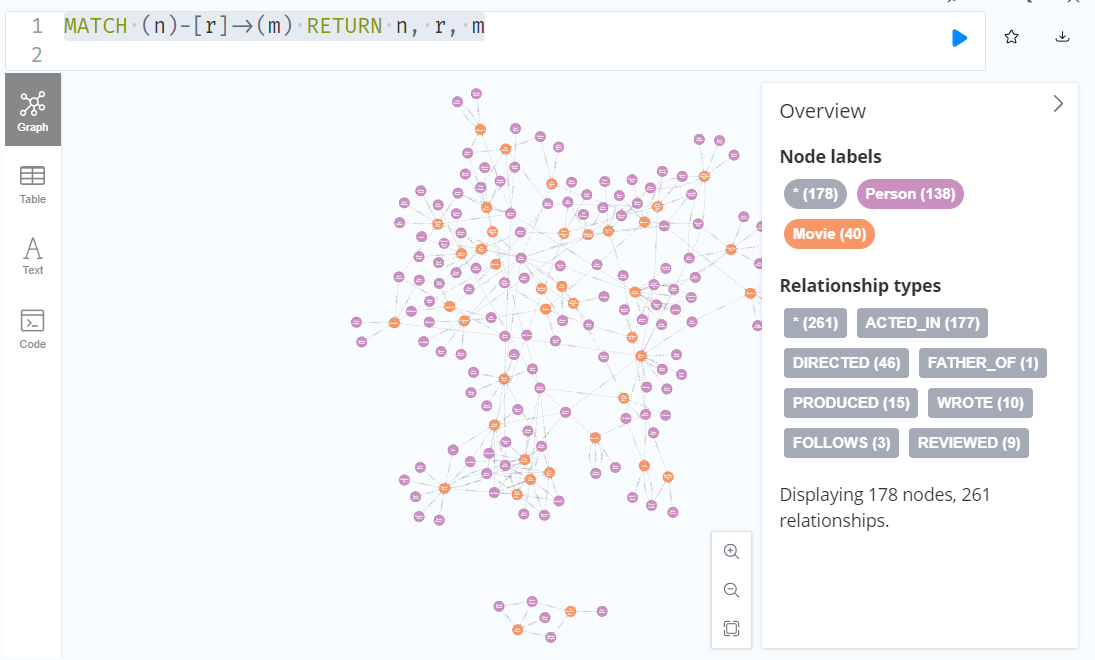
**DELETE r**



### 2.2.4. Lọc các nút.

#### 2.2.4.1. Hiển thị toàn bộ các nút và các mối quan hệ trước khi truy vấn

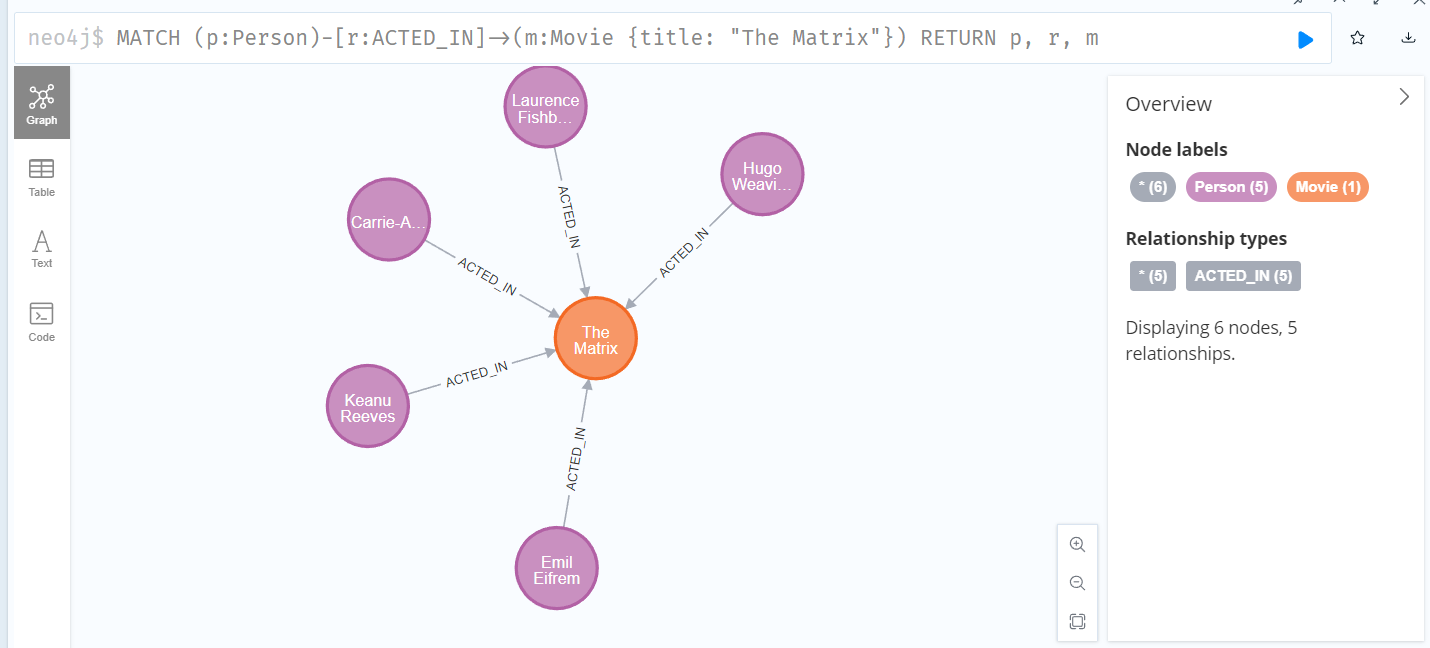
**MATCH (n)-[r]->(m) RETURN n, r, m**



#### 2.2.4.2. Truy vấn tất cả các người đã đóng một vai trong một bộ phim cụ thể:

**MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie {title: "The Matrix"})**

**RETURN p.name, r.roles, m.title**



Truy vấn này tìm tất cả các node Person có quan hệ ACTED\_IN với bộ phim "The Matrix" và trả về tên của họ cùng với vai diễn.

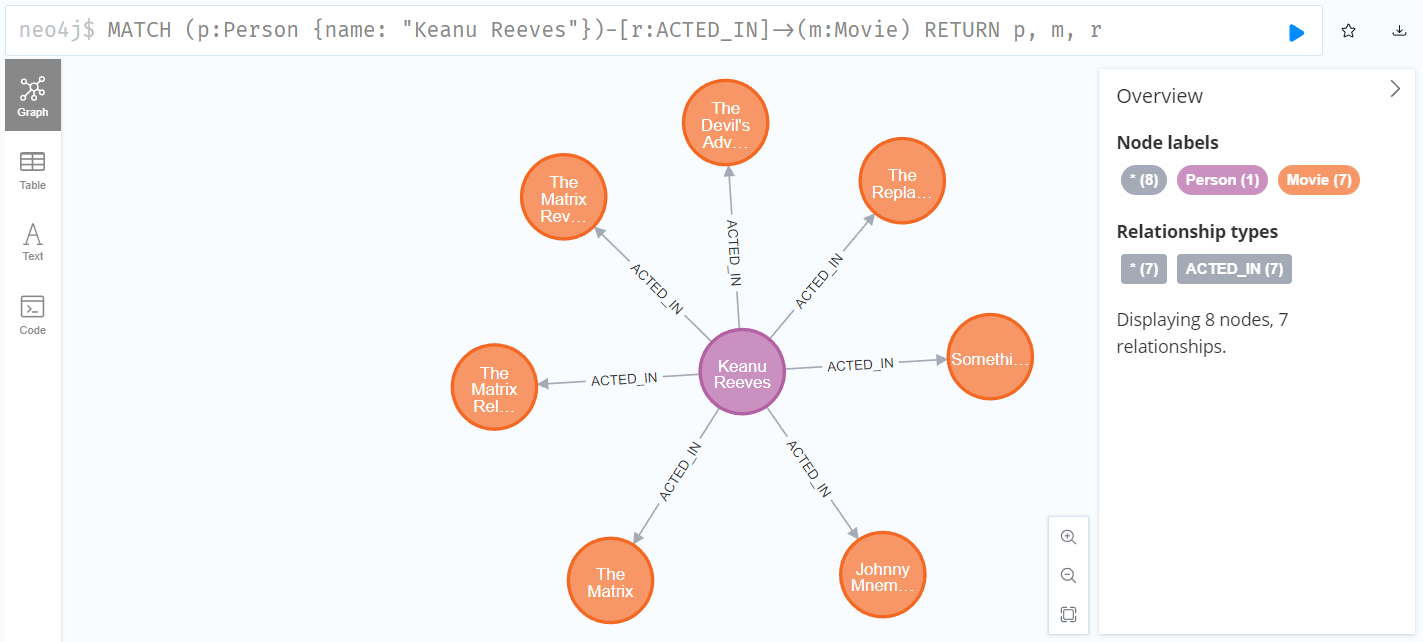
MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie {title: "The Matrix"}): Tìm mọi node là Person có quan hệ ACTED\_IN đến một node là Movie với tiêu đề "The Matrix".

RETURN p.name, r.roles, m.title: Trả về tên của diễn viên (p.name), vai trò của họ trong phim (r.roles) và tiêu đề của bộ phim (m.title).

#### 2.2.4.3. Truy vấn tất cả các bộ phim mà một diễn viên cụ thể đã tham gia:

**MATCH (p:Person {name: "Keanu Reeves"})-[r:ACTED\_IN]->(m:Movie)**

**RETURN p.name, m.title, r.roles**



Truy vấn này tìm tất cả các bộ phim mà "Keanu Reeves" đã đóng và trả về tên bộ phim cùng với vai diễn của anh ấy.

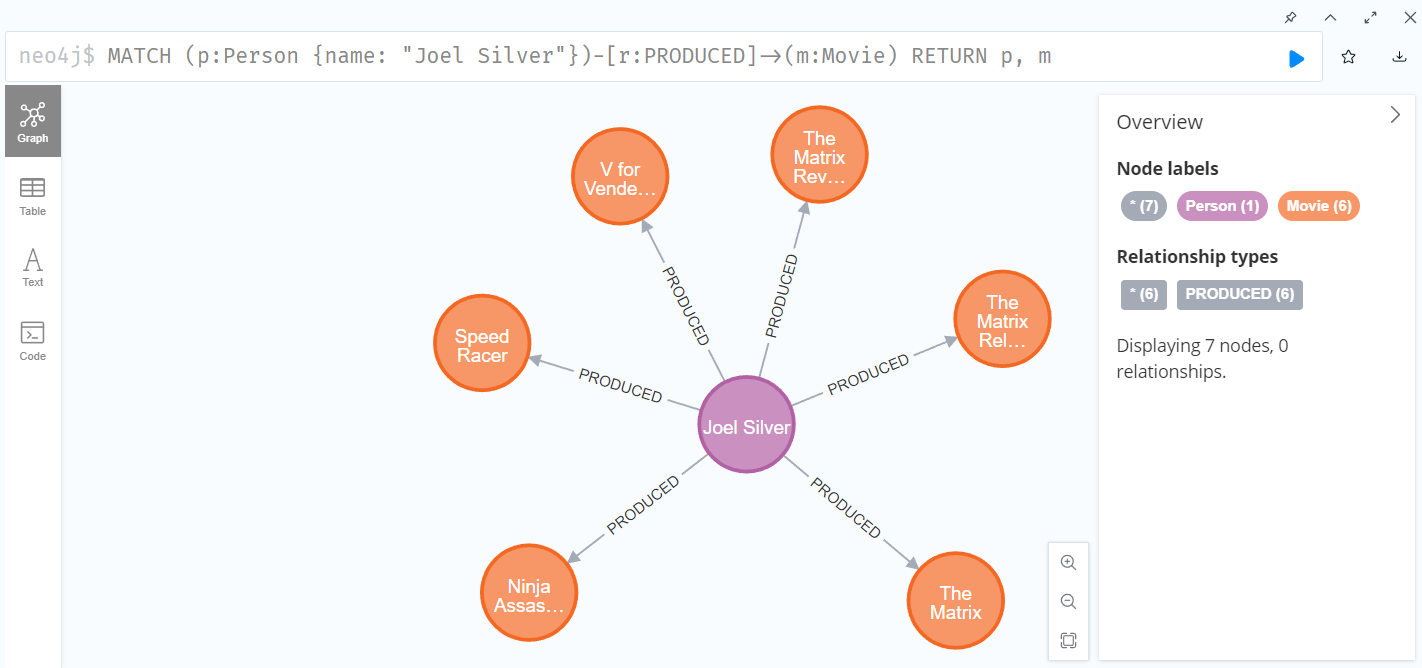
MATCH (p:Person {name: "Keanu Reeves"})-[r:ACTED\_IN]->(m:Movie): Tìm mọi node Movie mà Person tên là "Keanu Reeves" đã đóng.

RETURN p.name, m.title, r.roles: Trả về tên của diễn viên (p.name), tiêu đề của mỗi bộ phim (m.title) và vai trò của họ trong những phim đó (r.roles).

#### 2.2.4.4. Tìm tất cả các bộ phim được sản xuất bởi một người sản xuất cụ thể:

**MATCH (p:Person {name: "Joel Silver"})-[r:PRODUCED]->(m:Movie)**

**RETURN p.name, m.title**

****

Truy vấn này tìm tất cả các bộ phim được sản xuất bởi "Joel Silver".

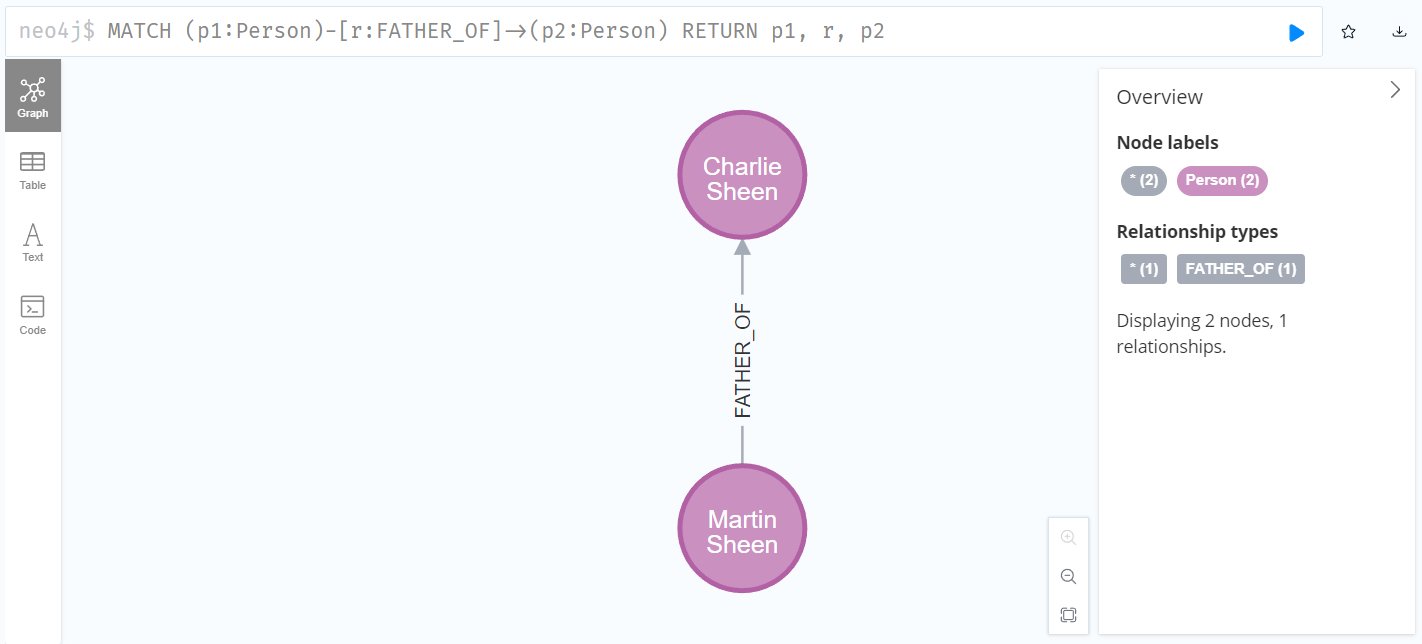
MATCH (p:Person {name: "Joel Silver"})-[r:PRODUCED]->(m:Movie): Tìm mọi node Movie mà Person tên là "Joel Silver" đã sản xuất.

RETURN p.name, m.title: Trả về tên của nhà sản xuất (p.name) và tiêu đề của các bộ phim mà họ đã sản xuất (m.title).

#### 2.2.4.5. Tìm tất cả các mối quan hệ gia đình:

**MATCH (p1:Person)-[r:FATHER\_OF]->(p2:Person)**

**RETURN p1.name, r, p2.name**

****

Truy vấn này tìm tất cả các quan hệ cha-con trong cơ sở dữ liệu.

MATCH (p1:Person)-[r:FATHER\_OF]->(p2:Person): Tìm mọi quan hệ FATHER\_OF giữa hai node Person.

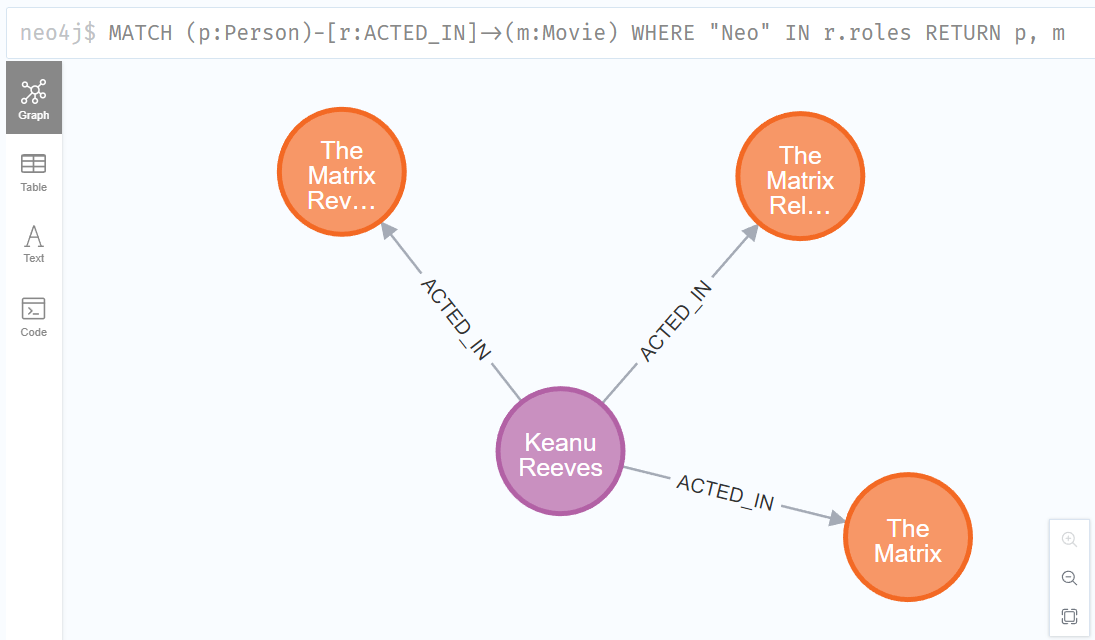
RETURN p1.name, r, p2.name: Trả về tên của cha (p1.name), loại quan hệ (r) và tên của con (p2.name).

#### 2.2.4.6. Tìm tất cả các người đã đóng vai chính trong một bộ phim:

**MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)**

**WHERE "Neo" IN r.roles**

**RETURN p.name, m.title**

****

Truy vấn này tìm tất cả các diễn viên đã đóng vai "Neo" trong các bộ phim.

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie): Tìm mọi node Person có quan hệ ACTED\_IN với một node Movie.

WHERE "Neo" IN r.roles: Lọc những quan hệ ACTED\_IN nơi vai trò bao gồm "Neo".

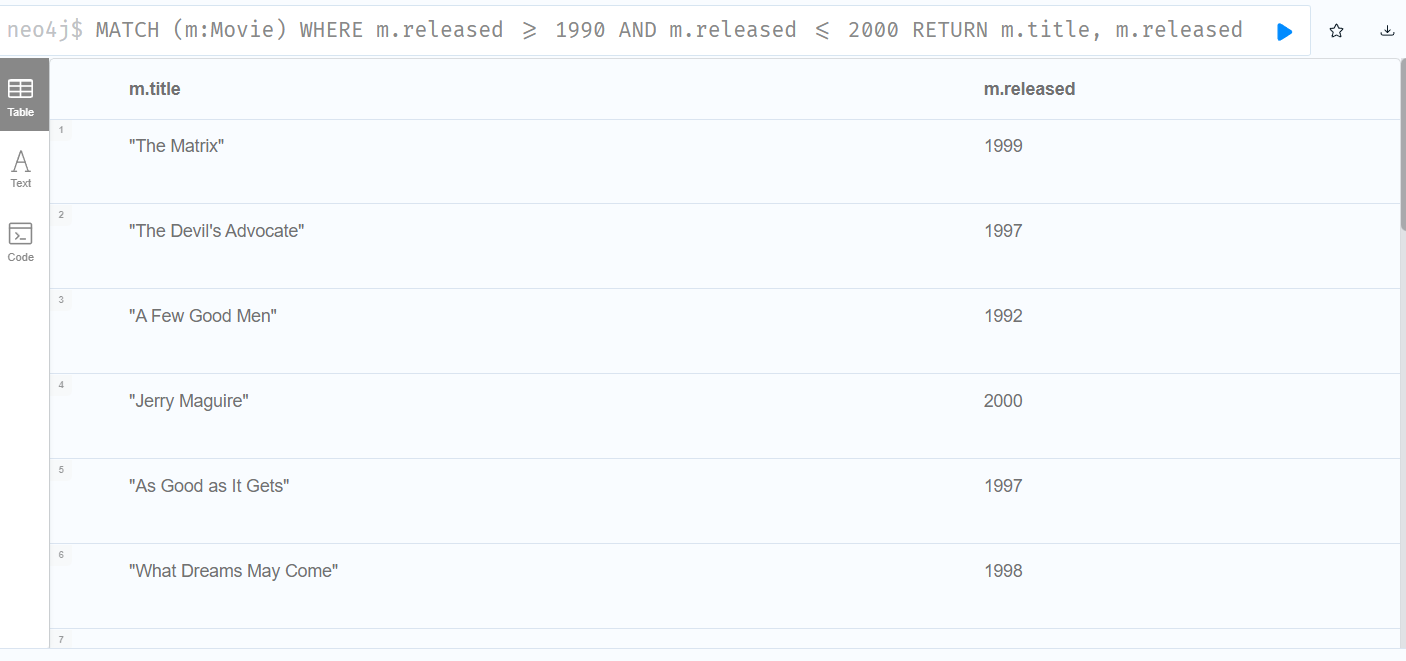
RETURN p.name, m.title: Trả về tên của diễn viên (p.name) và tiêu đề của bộ phim (m.title).

#### 2.2.4.7. Lọc các bộ phim theo năm phát hành:

**MATCH (m:Movie)**

**WHERE m.released >= 1990 AND m.released <= 2000**

**RETURN m.title, m.released**

****

Truy vấn này trả về tất cả các bộ phim được phát hành từ năm 1990 đến 2000.

MATCH (m:Movie): Tìm tất cả các node Movie.

WHERE m.released >= 1990 AND m.released <= 2000: Lọc các bộ phim phát hành từ năm 1990 đến 2000.

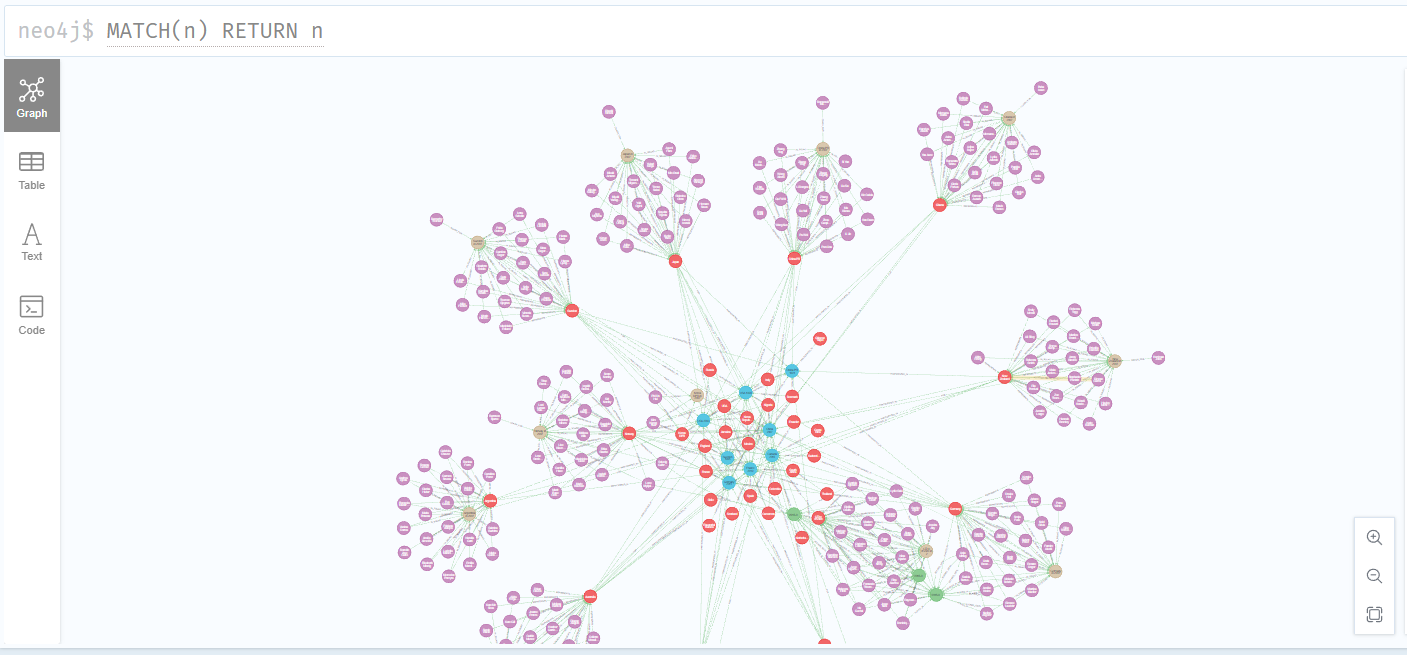
RETURN m.title, m.released: Trả về tiêu đề (m.title) và năm phát hành (m.released) của các bộ phim.

### 2.2.5. Truy vấn và lọc trên dữ liệu có sẵn

#### 2.2.5.1. Truy vấn các nút

Để hiển thị tất cả các node của DB ta thực thi lệnh :

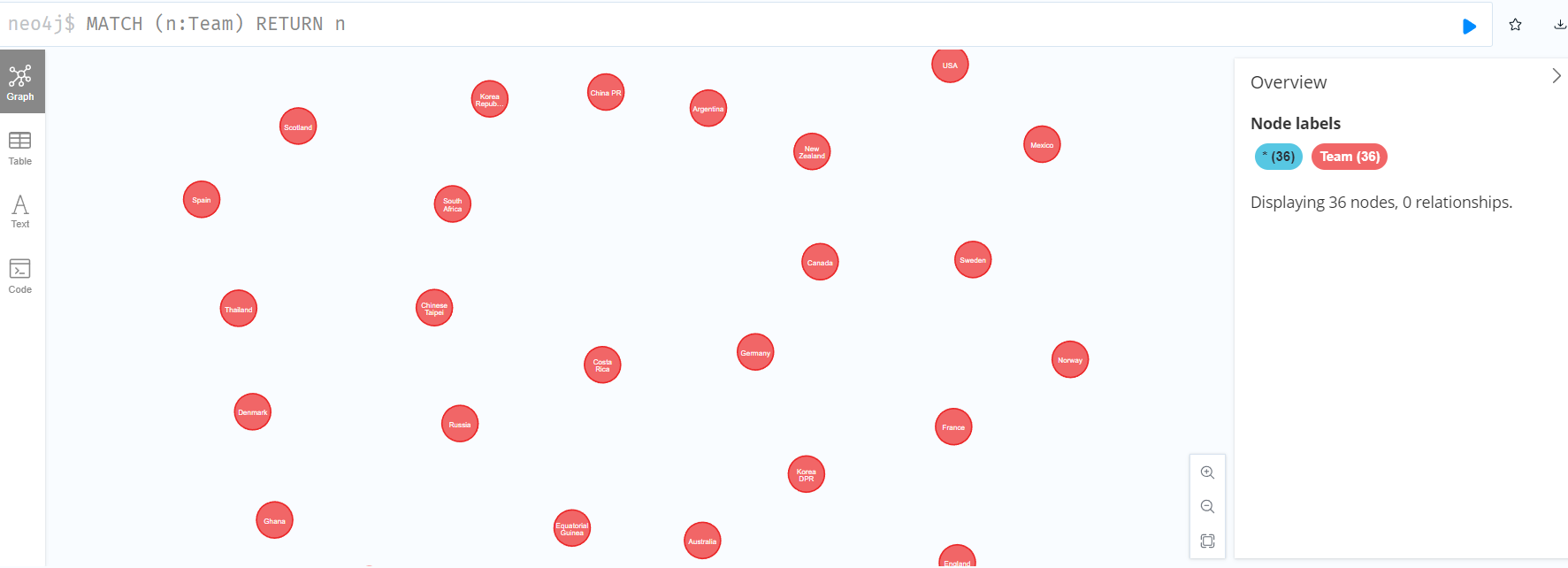
**MATCH ( n ) RETURN n**



Để hiển thị tất cả các team ta thực hiện câu lệnh:

**MATCH( n : Team ) RETURN n**

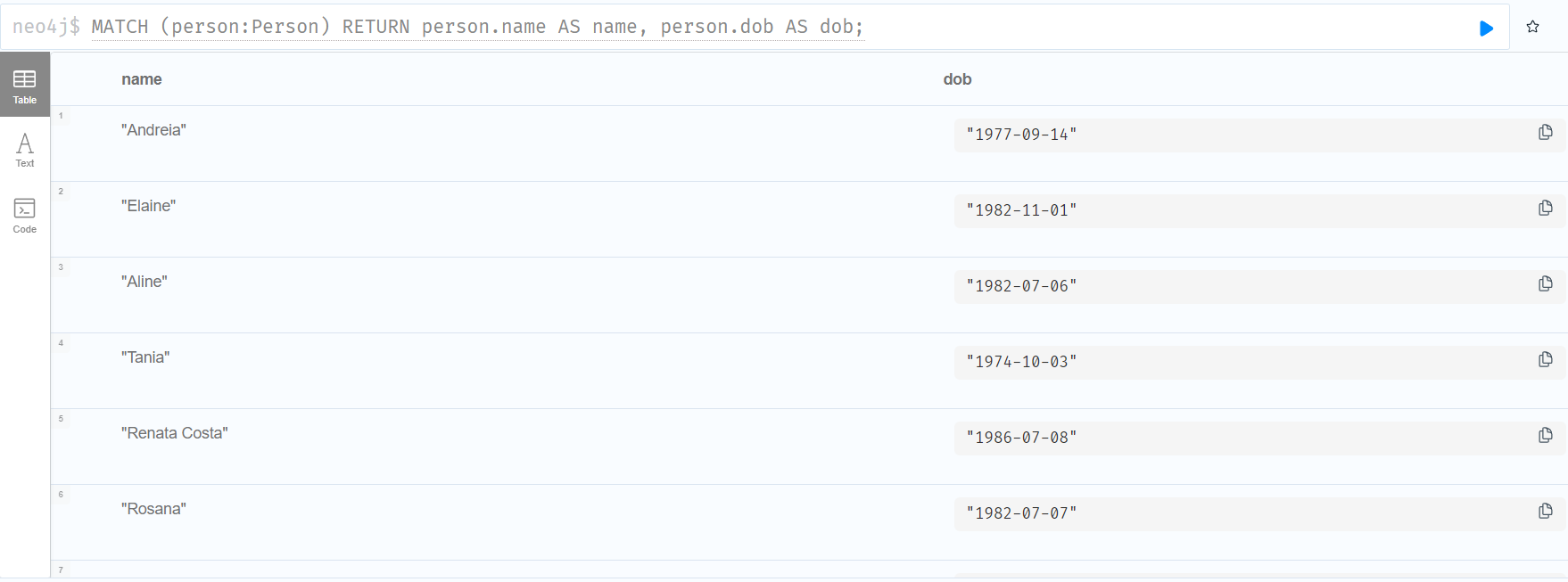
Gán tất cả các Node có nhãn là Team cho biến n , sau đó trả về n để hiển thị ra tất cả các Team .



Để hiển thị ra tên các Person và dob ( date of birth ) ta thực hiện câu lệnh:

**MATCH ( person : Person ) RETURN person.name AS name, person.dob AS dob;**

Lệnh này gán các node có nhãn Person cho biến person sau đó trả về hai properties của nhãn Person đó là name( person.name ) và dob ( person.dob ) . Kết quả sẽ hiển thị 2 cột name và dob.



Lọc những Person có name là Monica ta thực hiện:

**MATCH ( person:Person ) WHERE person.name = ‘Monica’ RETURN person.**

Lệnh này sẽ gán những node có label Person cho biến person , sau đó dùng WHERE để lọc điều kiện tìm ra những Person có thuộc tính name = ‘Monica’ ( person.name = ‘Monica’) , cuối cùng return lại person để hiển thị tất cả các node thỏa điều kiện trên .



Để lọc chi tiết hơn để tìm ra kết quả mong muốn ta thực hiện câu lệnh :

**MATCH ( person : Person ) WHERE person.name = ‘ Monica‘ AND person.id = ‘ 178664‘ RETURN person.**

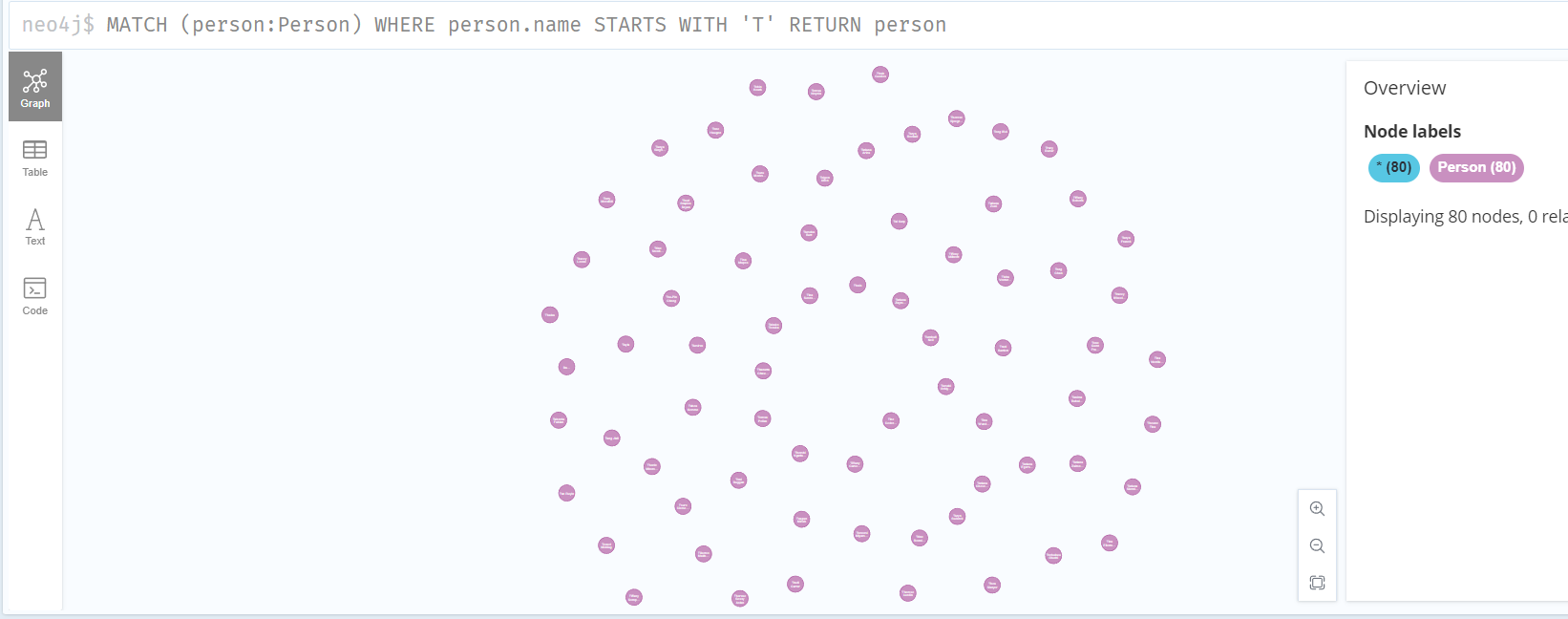
Ở lệnh này ta thêm vào ở lệnh WHERE để lọc chi tiết hơn nhằm tìm ra kết quả mong muốn , WHERE person.name = ‘ Monica‘ AND person.id = ‘ 178664‘ sẽ lọc và tìm ra Person thỏa cả 2 điều kiện trên , sau đó hiện thị ra kết quả thông qua lệnh return .



Lọc những Person có tên ( name ) bắt đầu bằng chữ T , ta thực hiện lệnh :

**MATCH ( person:Person ) WHERE person.name STARTS WITH ‘T’ RETURN person**

Điều kiện để lọc ở đây sẽ thay đổi bằng WHERE person.name STARTS WITH ‘T’ , lọc những Person có name bắt đầu bằng kí tự T .

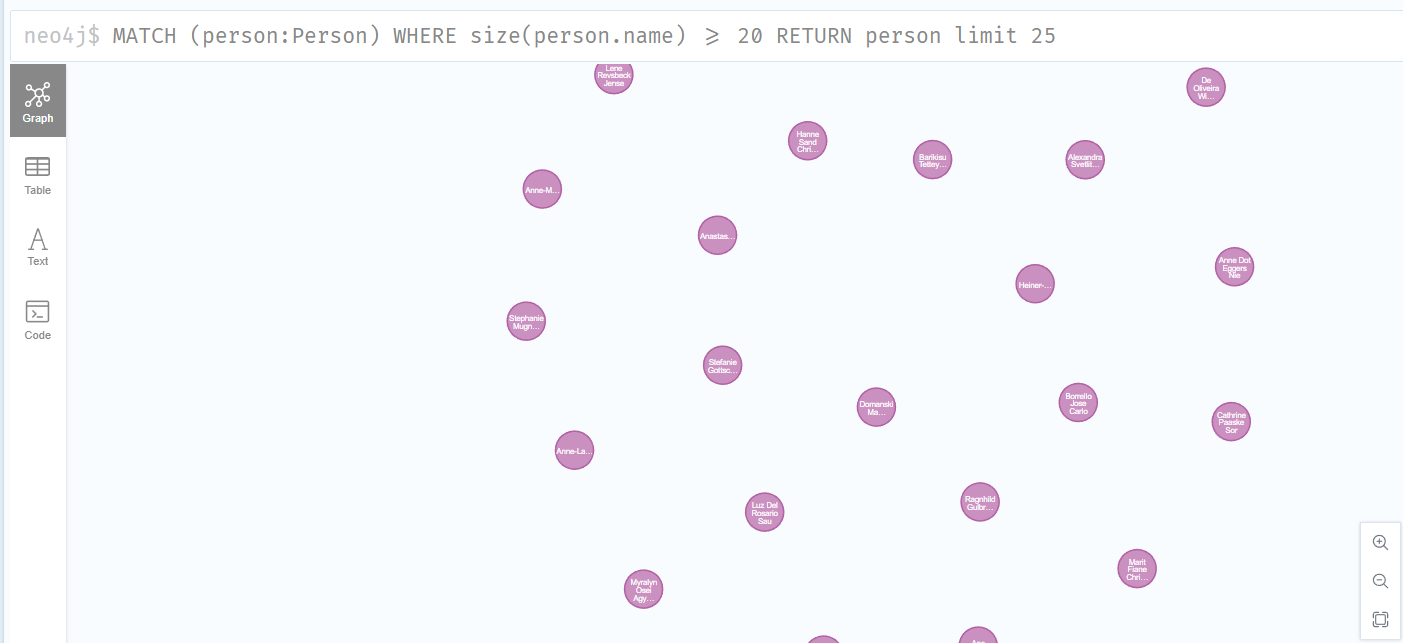


#### 2.2.5.2. Lọc các Node có tên cụ thể

Lọc những Person có độ dài của tên ( Person.name ) lớn hơn 25 , thực hiện lệnh:

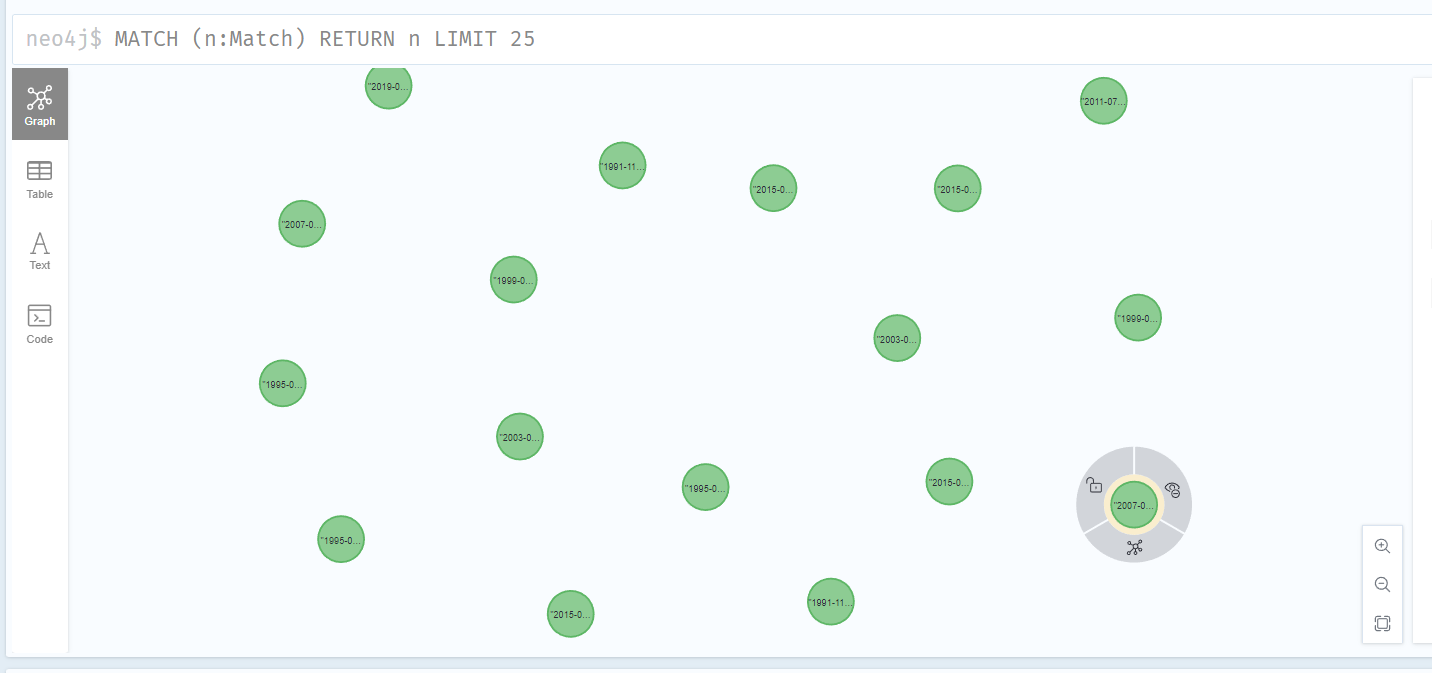
**MATCH ( person : Person ) WHERE size ( person.name ) >= 20 RETURN person LIMIT 25**

Điều kiện WHERE sẽ lọc những Person có độ dài của thuộc tính name lớn hơn hoặc bằng 20 , sau đó hiển thị ra 25 kết quả đầu tiên tìm thấy ( LIMIT 25 )



Lọc tất cả các trận đấu ( match ) trong database , ta thực hiện lệnh:

MATCH ( n : Match ) RETURN n LIMIT 25 , lệnh sẽ trả về các node có nhãn là Match và hiển thị ra 25 kết quả đâu tiên tìm thấy ( LIMIT 25 )

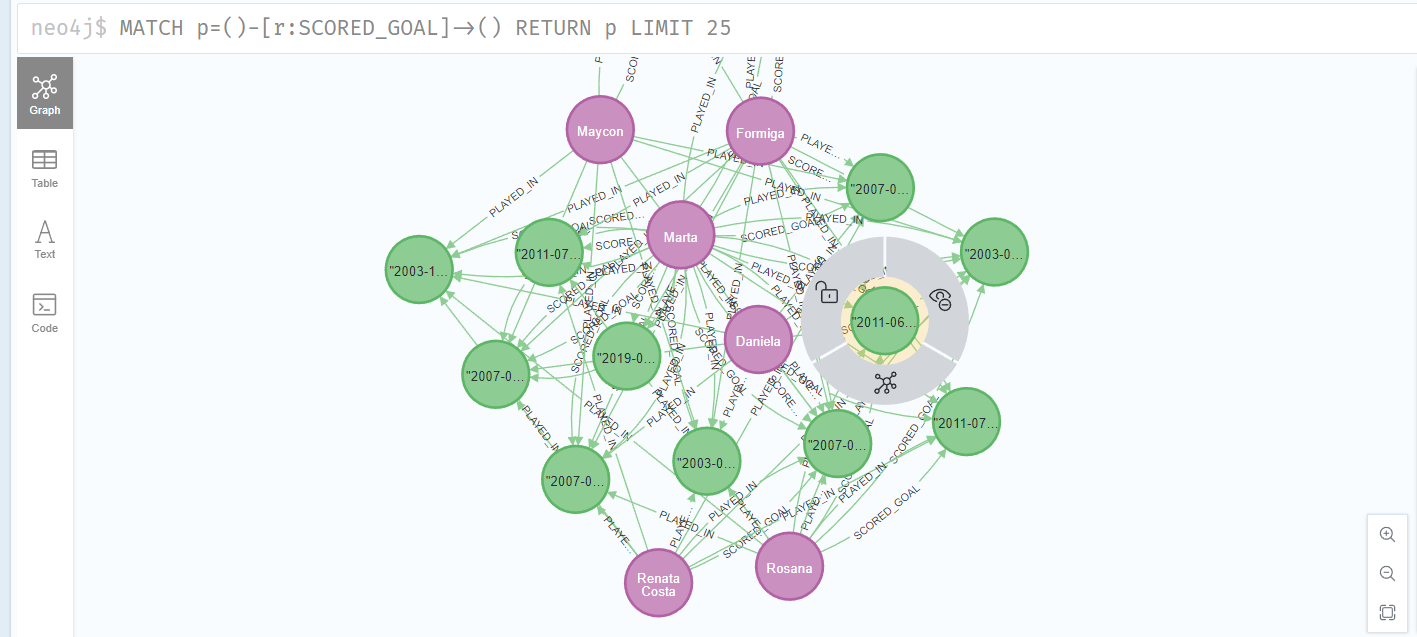


#### 2.2.5.3 Lọc và Truy vấn kết hợp các mối quan hệ .

Truy vấn các Node có mối quan hệ SCORED\_GOAL với nhau , ta thực hiện lệnh :

**MATCH p=( )-[r:SCORE\_GOAL] -> ( ) RETURN p LIMIT 25**

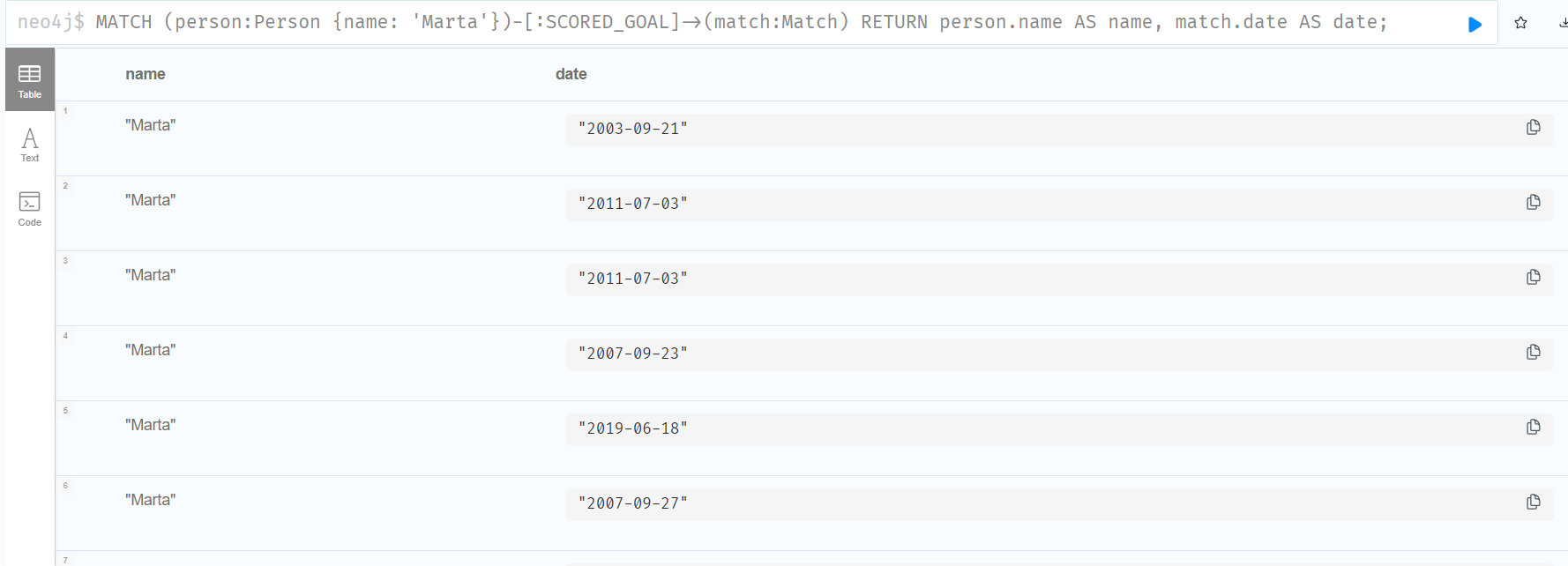
Lệnh này sẽ lấy tất cả các node có mối quan hệ SCORE\_GOAL ( [r:SCORE\_GOAL] ) và gán cho biến p , sau đó khi return p sẽ hiển thị ra và chỉ hiển thị 25 node ( RETURN p LIMIT 25 )



Lọc Person có tên là Marta đã ghi bàn ( mối quan hệ SCORED\_GOAL ) trong trận đấu ( match ) vào ngày tháng năm nào , ta thực hiện lệnh :

**MATCH ( person : Person {name: ‘Marta’ } )-[:SCORE\_GOAL] -> ( match:Match ) RETURN person.name AS name , match.date AS date ;**

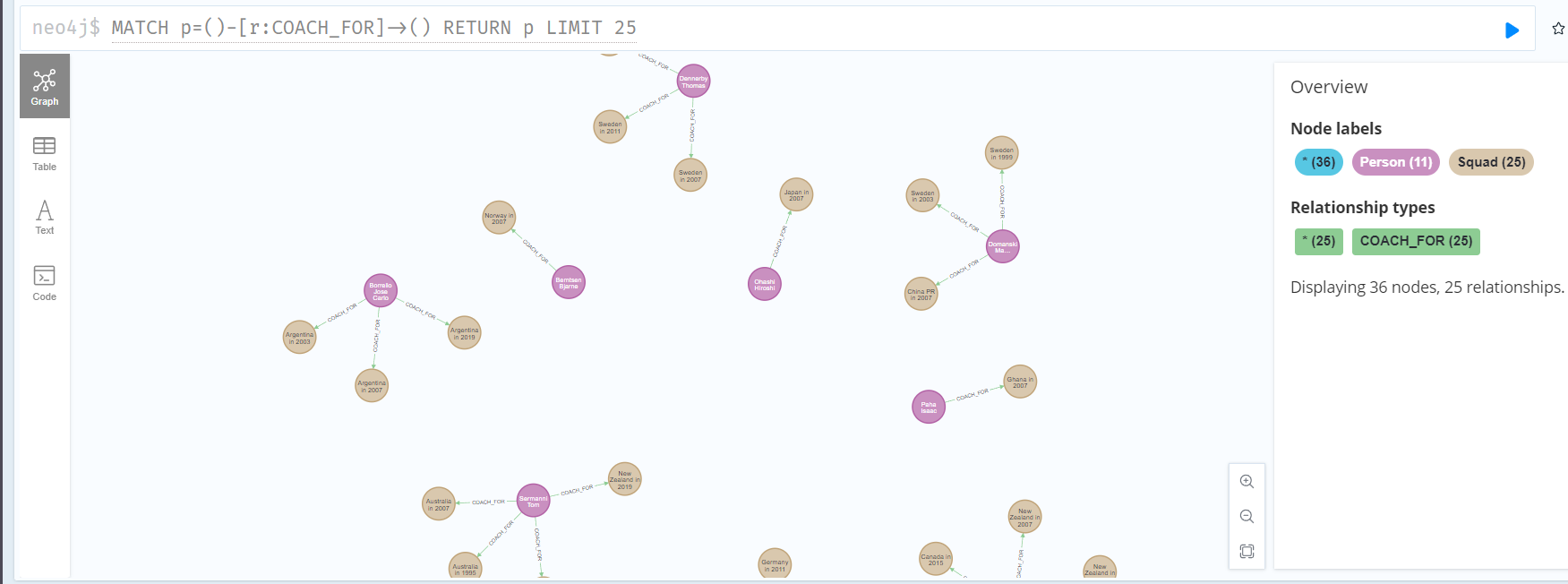
Lệnh này sẽ thực hiện gán những node có nhãn Person với name là Marta và có mối quan hệ SCORE\_GOAL với node có nhãn là Match cho biến person , sau đó trả về hai properties của mỗi node là name của Person và date của Match . Kết quả của lệnh sẽ cho hai cột với tên là name và date thể hiện ý nghĩa là Marta đã ghi bàn vào trận đấu nào ( ngày tháng năm của trận đấu ) .



Truy vấn các node có mối quan hệ COACH\_FOR với nhau ( Person từng là coach của những Squad nào?) , ta thực hiện lệnh :

**MATCH p = ( ) – [ r : COACH\_FOR ] -> RETURN p LIMIT 25**

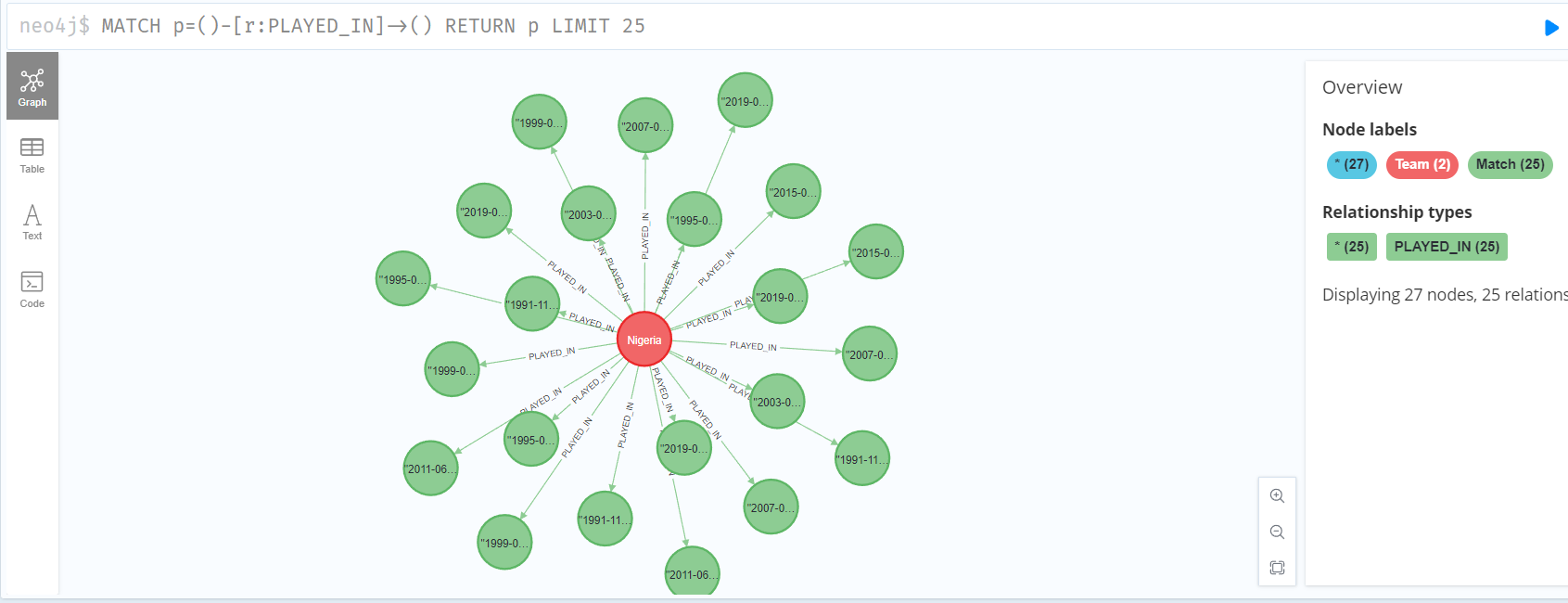
Lệnh này sẽ gán những node có mối quan hệ COACH\_FOR cho p và trả về 25 kết quả đầu tiền tìm được .



Truy vấn các node có mối quan hệ PLAYED\_IN với nhau ( Team đã chơi trong những trận đấu ( match ) nào ) , ta thực hiện lệnh:

**MATCH p = ( ) – [r : PLAYED\_IN] -> ( ) RETURN p LIMIT 25**

Lệnh sẽ thực hiện trả về 25 node có mối quan hệ PLAYED\_IN . Kết quả cho thấy ý nghĩa của mối quan hệ này .



# PHẦN 3: KẾT LUẬN

## 1. Kết quả đạt được

Nắm bắt được các kiến thức cũng như những vấn đề liên quan trọng về Neo4j, áp dụng kiến thức để thiết kế và phân tích một dữ liệu cụ thể. Biết được cách Neo4j hoạt động, đặc trưng, kiến trúc, cách tổ chức dữ liệu trong Neo4j.

## 2. Về demo

Neo4j: Sử dụng được các hàm cơ bản, truy vấn dữ liệu ổn, kết quả khá chính xác và có tính trực quan

## 3. Hạn chế

Neo4j: Còn tương đối hạn chế, chưa áp dụng được xử lý bằng các công cụ bổ sung như data science.

## 4. Hướng phát triển

Nếu có thể phát triển thêm, nhóm dự định sẽ sử dụng thêm các công cụ tính toán có sẵn của Neo4j để phát triển thêm mô hình dự đoán recommendation movie.

# TÀI LIỆU THAM KHẢO

References

1. Giới thiệu về NoSQL database. (n.d.). Viblo. Retrieved December 3, 2023, from https://viblo.asia/p/gioi-thieu-ve-nosql-database-djeZ1a9jZWz
2. NoSQL là gì? | Cơ sở dữ liệu phi quan hệ, Mô hình dữ liệu sơ đồ linh hoạt. (n.d.).

AWS. Retrieved December 3, 2023, from https://aws.amazon.com/vi/nosql/

1. Personalized Product Recommendations with Neo4j. (n.d.). Neo4j Browser Guides. Retrieved December 3, 2023, from https://guides.neo4j.com/sandbox/recommendations/index.html