

ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

---



# KỸ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN BÁO CÁO

NHÓM 2

CHỦ ĐỀ: GRAPH ANALYTICS WITH CHAT DATA USING  
NEO4J

**Giảng viên:** TS. Trần Hồng Việt

ThS. Ngô Minh Hương

**Sinh viên:** Khổng Ngọc Anh - 22022549

Phạm Công Đức - 22022607

Nguyễn Minh Đức - 21021292

Nguyễn Minh Hương - 22022542

Đỗ Ngọc Anh - 22022577

# Lời mở đầu

Trong kỷ nguyên số hóa, công nghệ thông tin đã và đang đóng vai trò cốt lõi trong việc thay đổi cách con người sống, làm việc và giao tiếp. Sự phát triển vượt bậc của các thiết bị thông minh, Internet, và các ứng dụng công nghệ đã tạo ra một khối lượng dữ liệu khổng lồ, phức tạp chưa từng có. Điển hình như các ứng dụng mạng xã hội, như Facebook có hơn 3 tỷ người dùng tích cực hàng tháng. Mỗi giây, những người dùng này tạo ra thêm hàng trăm triệu dữ liệu, bao gồm hình ảnh, văn bản,... Như vậy, nhu cầu phân tích và tận dụng dữ liệu để tạo ra giá trị trong kinh doanh, quản lý và đời sống ngày càng trở nên cấp thiết. Trong bối cảnh đó, Big Data nổi lên như một xu hướng không thể thiếu, mang lại khả năng khai thác sức mạnh của dữ liệu lớn để giải quyết các vấn đề thách thức và mở ra những cơ hội mới. Big Data không chỉ là một công nghệ, mà còn là chìa khóa giúp doanh nghiệp và tổ chức dẫn đầu trong thời đại cạnh tranh ngày nay.

Khi dữ liệu không chỉ đơn thuần là những con số mà còn mang tính chất kết nối phức tạp, việc quản lý và phân tích mối quan hệ giữa các dữ liệu trở thành yếu tố then chốt. Neo4j, một trong những hệ quản trị cơ sở dữ liệu đồ thị hàng đầu, đã nổi lên như một công cụ mạnh mẽ để giải quyết thách thức này. Với khả năng lưu trữ và truy vấn dữ liệu dưới dạng đồ thị, Neo4j cho phép các tổ chức phân tích sâu các mối quan hệ và cấu trúc mạng lưới một cách nhanh chóng và hiệu quả. Vì vậy, nhóm chúng em đã chọn "Graph analytics with chat data using Neo4j" làm chủ đề cho bài tập lớn lần này. Bản báo cáo của bọn em bao gồm các phần chính:

1. Tổng quan về BigData và dữ liệu Graph
2. Hệ quản trị cơ sở dữ liệu đồ thị Neo4j
3. Phân tích dữ liệu hội thoại bằng Neo4j

Bảng 1: PHÂN CHIA CÔNG VIỆC

Họ và Tên	Công việc
Khổng Ngọc Anh	+ Phân công công việc cho thành viên nhóm + Tìm hiểu và viết phần 3. Phân tích dữ liệu hội thoại bằng Neo4j + Xem xét, chỉnh sửa lại báo cáo
Phạm Công Đức	+ Tìm hiểu và viết phần 3. Phân tích dữ liệu hội thoại bằng Neo4j + Cài đặt phân tích dữ liệu bằng Neo4j + Làm slide
Nguyễn Minh Hường	+ Tìm hiểu và viết phần 2. Hệ quản trị cơ sở dữ liệu đồ thị Neo4j + Làm slide
Đỗ Ngọc Anh	+ Tìm hiểu và viết phần 2. Hệ quản trị cơ sở dữ liệu đồ thị Neo4j + Làm slide
Nguyễn Minh Đức	+ Tìm hiểu và viết phần 1. Tổng quan về BigData và dữ liệu Graph + Làm slide

## 0.1 Đánh giá công việc hoàn thành

*Trong bài tập lớn này tất cả mọi thành viên của nhóm đều hoàn thành công việc được giao, công việc đều được chia đều và mọi thành viên nhóm đóng góp cho bài tập là như nhau, nhóm trưởng (Khổng Ngọc Anh) có thêm nhiệm vụ phân công công việc cho các thành viên*

# Mục lục

0.1	Đánh giá công việc hoàn thành . . . . .	2
<b>1</b>	<b>Tổng quan về BigData và dữ liệu Graph</b>	<b>4</b>
1.1	Giới thiệu về BigData . . . . .	4
1.1.1	Định nghĩa . . . . .	4
1.1.2	Phân loại . . . . .	4
1.2	Dữ liệu dạng Graph và Graph database . . . . .	5
1.2.1	Định nghĩa . . . . .	5
1.3	Giới thiệu về Graph Database . . . . .	5
1.3.1	Định nghĩa . . . . .	5
1.3.2	Ứng dụng của cơ sở dữ liệu đồ thị . . . . .	5
1.3.3	Cách hoạt động . . . . .	6
<b>2</b>	<b>Hệ quản trị cơ sở dữ liệu đồ thị Neo4j</b>	<b>7</b>
2.1	Tổng quan về Neo4j . . . . .	7
2.1.1	Mô hình đồ thị . . . . .	7
2.1.2	Tính năng của Neo4j . . . . .	8
2.1.3	Hệ sinh thái Neo4j . . . . .	8
2.2	Ngôn ngữ truy vấn Cypher . . . . .	10
2.2.1	Giới thiệu . . . . .	10
2.2.2	Truy vấn trong Cypher . . . . .	10
2.2.3	Một vài truy vấn cơ bản . . . . .	11
<b>3</b>	<b>Phân tích dữ liệu hội thoại bằng Neo4j</b>	<b>13</b>
3.1	Tổng quan bộ dữ liệu . . . . .	13
3.1.1	Cấu trúc bộ dữ liệu . . . . .	13
3.1.2	Đặc điểm nổi bật . . . . .	13
3.1.3	Ứng dụng tiềm năng . . . . .	14
3.2	Mô hình hóa bộ dữ liệu . . . . .	14
3.2.1	Mô hình hóa cấu trúc đồ thị . . . . .	14
3.2.2	Biểu diễn mối quan hệ trong đồ thị . . . . .	15
3.2.3	Tiến hành nhập dữ liệu . . . . .	16
3.3	Trực quan hóa và phân tích dữ liệu . . . . .	17
3.3.1	Các truy vấn cơ bản . . . . .	17
3.3.2	Phân tích nâng cao với Graph Data Science . . . . .	18

# Chương 1

## Tổng quan về BigData và dữ liệu Graph

### 1.1 Giới thiệu về BigData

#### 1.1.1 Định nghĩa

BigData là các tập dữ liệu rất lớn và phức tạp, rất khó để quản lý, lưu trữ và phân tích bằng các công cụ xử lý dữ liệu truyền thống. Điều quan trọng của big data là khả năng phân tích và tìm hiểu thông tin từ những tập dữ liệu này, vì chúng thường chứa nhiều thông tin tiềm ẩn và giá trị quan trọng.

BigData đặc trưng bởi 3 yếu tố chính, được gọi là "**3V**": Lượng dữ liệu lớn (**Volume**), tốc độ xử lý nhanh (**Velocity**) và tính đa dạng, linh hoạt (**Variety**). Trong đó:

- **Volume** là khối lượng dữ liệu được các doanh nghiệp thu thập từ các nguồn khác nhau, như các nguồn thiết bị điện tử, video, giao dịch, các phương tiện truyền thông.
- **Velocity** chỉ tốc độ mà dữ liệu mới được sinh ra, cũng chính là tốc độ ta cần phải xử lý chúng. Đặc biệt, với sự phát triển của các thiết bị điện tử thông minh, các luồng dữ liệu truyền tải với tốc độ cực nhanh và chúng phải được xử lý kịp thời.
- **Variety** chỉ tính đa dạng về mặt định dạng, cấu trúc, lĩnh vực của dữ liệu, bao gồm dữ liệu cấu trúc và phi cấu trúc, dữ liệu số, email, video, âm thanh, giao dịch tài chính,... Tính đa dạng ảnh hưởng đến hiệu suất, đây là một trong những vấn đề chính mà lĩnh vực Big data cần phải giải quyết.

#### 1.1.2 Phân loại

Dữ liệu trong BigData có thể chia thành các nhóm sau:

- **Dữ liệu có cấu trúc:** Dữ liệu có cấu trúc là loại dữ liệu dễ quản lý và tìm kiếm nhất, vì nó có thể được truy cập, lưu trữ và xử lý trong một định dạng cố định. Các thành phần của dữ liệu này được phân loại rõ ràng, giúp các nhà thiết kế và quản trị viên cơ sở dữ liệu dễ dàng xác định các thuật toán đơn giản để thực hiện việc tìm kiếm và phân tích.
- **Dữ liệu phi cấu trúc:** Dữ liệu phi cấu trúc là bất kỳ tập dữ liệu nào không được tổ chức hoặc xác định một cách rõ ràng. Loại dữ liệu này thường hỗn loạn, khó xử lý, hiểu và đánh giá. Nó không có một cấu trúc cố định và có thể thay đổi theo thời gian. Dữ liệu phi cấu trúc bao gồm các nhận xét, tweet, lượt chia sẻ, bài đăng trên mạng xã hội, video trên YouTube mà người dùng xem, và nhiều dạng dữ liệu khác.

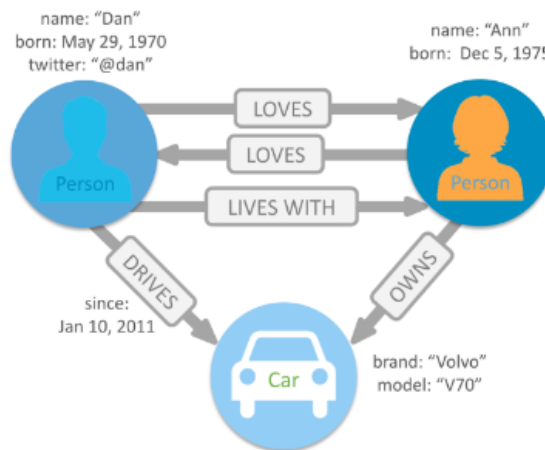
- **Dữ liệu bán cấu trúc:** Dữ liệu bán cấu trúc là sự pha trộn giữa dữ liệu có cấu trúc và dữ liệu phi cấu trúc. Một ví dụ điển hình là email, trong đó nội dung thư là dữ liệu phi cấu trúc, trong khi các thuộc tính tổ chức như người gửi, người nhận, chủ đề và ngày tháng lại là dữ liệu có cấu trúc. Các thiết bị sử dụng gắn thẻ địa lý và thời gian cũng có thể cung cấp dữ liệu có cấu trúc bên cạnh nội dung phi cấu trúc.

## 1.2 Dữ liệu dạng Graph và Graph database

### 1.2.1 Định nghĩa

Đồ thị là dạng các dữ liệu được kết nối với nhau. Các thành phần của đồ thị bao gồm:

- **Nodes:** Biểu diễn các chủ thể trong đồ thị, có thể dẫn nhãn
- **Relationships:** Các cạnh trong đồ thị biểu hiện mối quan hệ giữa các chủ thể trong đồ thị
- **Properties:** Các cặp name-value có thể được gán cho nodes và relationships



Hình 1.1: Hình ảnh minh họa một đồ thị

## 1.3 Giới thiệu về Graph Database

### 1.3.1 Định nghĩa

Cơ sở dữ liệu đồ thị (Graph Database) là loại cơ sở dữ liệu được thiết kế để coi trọng các mối quan hệ giữa các dữ liệu, ngang bằng với chính dữ liệu đó. Nó được xây dựng để lưu trữ dữ liệu mà không cần chuyển đổi nó thành một mô hình cố định. Thay vào đó, dữ liệu được lưu trữ theo cách giống như khi chúng ta vẽ ra lần đầu tiên, cho thấy cách mỗi thực thể kết nối hoặc liên quan với các thực thể khác.

### 1.3.2 Ứng dụng của cơ sở dữ liệu đồ thị

Chỉ những cơ sở dữ liệu lưu trữ các mối quan hệ nguyên bản mới có thể xử lý và truy vấn các kết nối một cách hiệu quả. Trong khi các cơ sở dữ liệu khác phải tính toán các mối quan hệ khi truy vấn thông qua các thao tác JOIN tốn thời gian, cơ sở dữ liệu đồ thị lại lưu trữ các kết nối trực tiếp với dữ liệu trong mô hình của nó. Việc truy cập các

nút và mối quan hệ trong cơ sở dữ liệu đồ thị gốc diễn ra nhanh chóng và hiệu quả, cho phép duyệt qua hàng triệu kết nối mỗi giây trên mỗi lõi. Bất kể kích thước tổng thể của dữ liệu, cơ sở dữ liệu đồ thị vẫn vượt trội trong việc quản lý dữ liệu có tính kết nối cao và các truy vấn phức tạp. Chỉ với một mẫu và một tập hợp các điểm bắt đầu, cơ sở dữ liệu đồ thị có thể khám phá các dữ liệu lân cận quanh các điểm xuất phát, thu thập và tổng hợp thông tin từ hàng triệu nút và mối quan hệ, trong khi dữ liệu ngoài phạm vi tìm kiếm không bị ảnh hưởng.

### 1.3.3 Cách hoạt động

Cơ sở dữ liệu đồ thị thường đơn giản về cách chúng được cấu trúc. Chúng chủ yếu bao gồm hai thành phần:

- **Nút (Node):** Một nút đại diện cho một thực thể. Các nút sẽ ghi nhận thông tin của thực thể
- **Cạnh (Edge):** Biểu diễn mối quan hệ giữa hai nút. Các cạnh cũng có thể có các phần thông tin riêng của chúng, chẳng hạn như bản chất của mối quan hệ giữa hai nút. Tương tự, các cạnh cũng có thể có các hướng mô tả luồng dữ liệu đã nói.

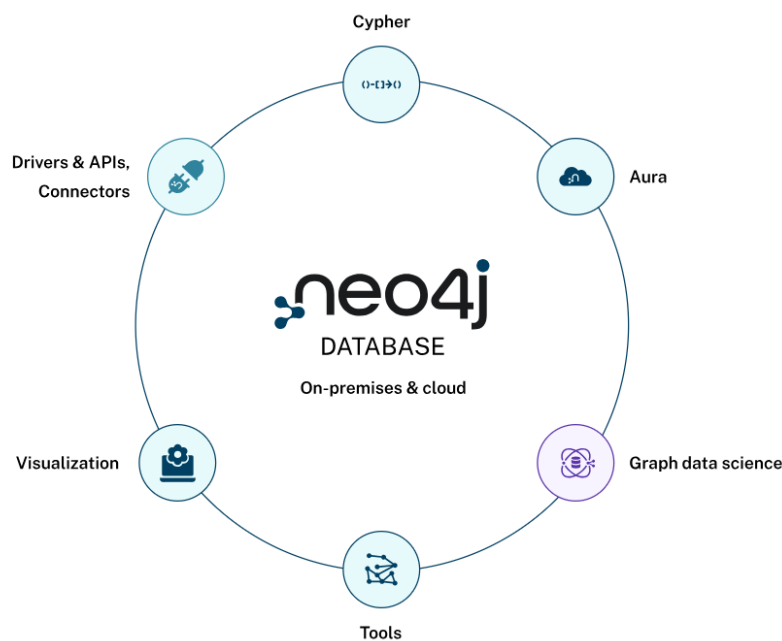
Thông tin được sử dụng trong cơ sở dữ liệu đồ thị về cơ bản có thể là bất cứ thứ gì, và khá đơn giản để vẽ ra và hiểu ở mức cơ bản. Trên thực tế, rất nhiều cơ sở dữ liệu đồ thị hiện đại đang bắt đầu bao gồm loại hình ảnh trực quan nhanh này không yêu cầu kiến thức ngôn ngữ cơ sở dữ liệu phức tạp.

## Chương 2

# Hệ quản trị cơ sở dữ liệu đồ thị Neo4j

### 2.1 Tổng quan về Neo4j

Neo4j là 1 cơ sở dữ liệu đồ thị gốc (native graph database), nghĩa là nó được thiết kế và tối ưu hóa cụ thể để lưu trữ và truy vấn dữ liệu dưới dạng đồ thị ngay từ đầu. Điều này cho phép Neo4j cung cấp khả năng vượt trội trong việc biểu diễn và xử lý dữ liệu kết nối (connected data) so với các cơ sở dữ liệu truyền thống cần phải sử dụng một bước "trừu tượng hóa đồ thị" (graph abstraction) trên các cơ chế lưu trữ không phải đồ thị.



Hình 2.1: Neo4j

#### 2.1.1 Mô hình đồ thị

Neo4j triển khai một mô hình đồ thị ở mọi cấp độ [1]:

- Tầng lưu trữ (Storage Layer): Dữ liệu được lưu trữ dưới dạng các nodes và mối quan hệ, phản ánh đúng cách ta hình dung về nó.



- Công cụ duyệt (Traversal Engine): Được tối ưu hóa cho các thao tác trên đồ thị, như duyệt qua các mối quan hệ và tìm đường đi, loại bỏ nhu cầu sử dụng các phép nối (joins) và tính toán phức tạp.

Khi thiết kế với Neo4j, cách lập sơ đồ ý tưởng (ví dụ: mô hình ER) có thể được chuyển đổi trực tiếp thành cấu trúc của cơ sở dữ liệu.

- Nút (Nodes) đại diện cho các thực thể hoặc đối tượng trong cơ sở dữ liệu
  - Mỗi nút có thể chứa nhiều thuộc tính dưới dạng key-value
  - Các nút thường được gắn nhãn (Label) để phân loại
  - Một nút có thể có nhiều nhãn
- Quan hệ (Relationships): Mối quan hệ giữa các nút, biểu thị cách các nút tương tác với nhau
  - Được gắn nhãn để mô tả kiểu quan hệ
  - Mỗi mối quan hệ có một hướng cụ thể
  - Có thể chứa thuộc tính, giống như các nút
- Thuộc tính (Properties) bổ sung ngữ cảnh cho cả nút và mối quan hệ
  - Mỗi nút và quan hệ có thể chứa nhiều thuộc tính
  - Dữ liệu thuộc tính có thể là số, chuỗi, boolean, danh sách hoặc NULL

### 2.1.2 Tính năng của Neo4j

Neo4j cung cấp các tính năng khiến nó trở thành một lựa chọn mạnh mẽ, đáng tin cậy và linh hoạt cho các ứng dụng dựa trên đồ thị.

1. **Tuân thủ ACID:** Neo4j đảm bảo nguyên tắc ACID (Tính nguyên tử - Atomicity, Tính nhất quán - Consistency, Tính cô lập - Isolation, Tính bền vững - Durability)
2. **Tính phân cụm và tính khả dụng:** Neo4j hỗ trợ kiến trúc phân tán để xử lý dữ liệu quy mô lớn và lưu lượng truy cập cao. Đồng thời đảm bảo tính khả dụng bằng cách tự động chuyển hướng truy vấn đến các nút đang hoạt động trong cụm khi xảy ra lỗi
3. **Tối ưu hóa hiệu suất:** Thiết kế của Neo4j mang lại:
  - Duyệt nhanh: Các mối quan hệ được lưu trữ dưới dạng con trỏ trực tiếp
  - Tối ưu hóa tìm đường: Các thuật toán như tìm đường ngắn nhất, trung tâm (centrality) và phát hiện cộng đồng (community detection), giúp truy vấn được thực thi hiệu quả

### 2.1.3 Hệ sinh thái Neo4j

Neo4j có một hệ sinh thái với các công cụ đa dạng, bao gồm:

1. **Công cụ**
  - Neo4j Desktop: Ứng dụng cục bộ để quản lý cơ sở dữ liệu và chạy các truy vấn Cypher
  - Neo4j Browser: Giao diện thực hiện truy vấn và hiển thị dữ liệu

- Neo4j Aura: Dịch vụ đám mây giúp triển khai các phiên bản Neo4j
2. **Giải thuật đồ thị và khoa học dữ liệu:** Neo4j cung cấp các thuật toán cho phân tích đồ thị nâng cao thông qua thư viện Graph Data Science (GDS)
- Các thuật toán phát hiện cộng đồng, đo độ trung tâm (centrality), và đo độ tương đồng (similarity)
  - Tìm đường đi và xây dựng hệ thống gợi ý
3. **Ngôn ngữ truy vấn**
- Cypher: Ngôn ngữ truy vấn trực quan, tương tự SQL, được thiết kế riêng cho các thao tác trên đồ thị
  - Neo4j cũng hỗ trợ tích hợp với các ngôn ngữ truy vấn khác, bao gồm GraphQL
4. **Các thư viện**
- Hỗ trợ tương tác bằng các ngôn ngữ như Java, Python, JavaScript và .NET
  - Thư viện Neo4j GraphQL: Đơn giản hóa việc xây dựng API với graph-based backend logic
  - Neo4j Bloom: Công cụ trực quan để khám phá và tương tác với dữ liệu đồ thị mà không cần viết truy vấn
5. **Tích hợp với Big Data và ML**
- Apache Spark Connector: Để xử lý các tập dữ liệu lớn
  - Python and R Packages: Hỗ trợ quy trình làm việc trong học máy
  - Công cụ Nhập Dữ liệu (Data Import Tools) : Tích hợp với các cơ sở dữ liệu quan hệ, tệp CSV và hệ thống NoSQL

Hệ sinh thái Neo4j cung cấp nhiều lợi ích, bao gồm:

- **Tính linh hoạt (Flexibility):** Thiết kế không cần cấu trúc (schema-less) cho phép phát triển mô hình dữ liệu mà không làm gián đoạn hoạt động
- **Khả năng truy vấn phong phú (Rich Query Capabilities):** Cypher và tích hợp với GraphQL đơn giản hóa các truy vấn phức tạp
- **Khả năng mở rộng (Scalability):** Xử lý các tập dữ liệu ngày càng lớn với cụm (clustering), phân mảnh (sharding), và duyệt hiệu quả
- **Khả năng tương tác (Interoperability):** Kết nối với các công cụ, ngôn ngữ và nền tảng hiện có

Để tương tác với Neo4j, ta có thể sử dụng các phương pháp sau:

- **Ngôn ngữ truy vấn Cypher:** Phương pháp chính để truy vấn và quản lý cơ sở dữ liệu
- **Sử dụng các ngôn ngữ lập trình:** Java, Python, JavaScript, .NET, Go,...
- **GraphQL:** Dành cho việc xây dựng API sử dụng dữ liệu đồ thị một cách liền mạch
- **Các công cụ tương tác:** Sử dụng Neo4j Browser và Bloom để dễ dàng truy vấn và trực quan hóa dữ liệu

## 2.2 Ngôn ngữ truy vấn Cypher

### 2.2.1 Giới thiệu

Cypher là ngôn ngữ truy vấn của Neo4j, giúp người dùng khai thác tối đa tiềm năng của cơ sở dữ liệu đồ thị (Property Graph Database). Cypher là công cụ lý tưởng để khai thác và biểu diễn dữ liệu đồ thị, giúp người dùng dễ dàng phát hiện ra các thông tin giá trị trong cơ sở dữ liệu của mình. Chúng ta có thể dễ khám phá các kết nối dữ liệu ẩn và các cụm thông tin chưa được biết đến nhờ Cypher. Sử dụng Cypher cũng giống như đang "vẽ" một mẫu dữ liệu trực tiếp lên đồ thị điều này làm cho nó trở nên dễ hiểu và dễ viết.

Cypher được tạo ra vào năm 2011 bởi đội ngũ kỹ sư của Neo4j, với mục tiêu cung cấp một ngôn ngữ tương đương SQL cho cơ sở dữ liệu đồ thị. Tương tự SQL, Cypher cho phép người dùng tập trung vào việc "muốn lấy gì" từ đồ thị, thay vì "lấy như thế nào". Điều này giúp Cypher trở thành công cụ mạnh mẽ để thực hiện các truy vấn hiệu quả và rõ ràng. Điểm khác biệt chính giữa Cypher và SQL là:

- **Linh hoạt về schema:** Cypher và Neo4j không yêu cầu một schema cố định, cho phép người dùng thêm thuộc tính và mối quan hệ mới khi dữ liệu phát triển. Điều này mang lại sự linh hoạt hơn nhiều so với SQL và cơ sở dữ liệu quan hệ.
- **Thứ tự truy vấn:** SQL bắt đầu với câu lệnh trả về dữ liệu (SELECT) trong khi Cypher câu lệnh RETURN (trả về dữ liệu) nằm ở cuối cùng.
- **Truy vấn ngắn gọn hơn:** Cú pháp của Cypher thường súc tích hơn SQL nhờ cách biểu diễn trực quan giống như bảng vẽ.

### 2.2.2 Truy vấn trong Cypher

Về cơ bản, cơ sở dữ liệu đồ thị Neo4j bao gồm ba thực thể cốt lõi: nút (nodes), mối quan hệ (relationships) và đường dẫn (paths) [2]. Các ví dụ dưới đây sử dụng mệnh đề MATCH và RETURN để tìm và trả về các mẫu biểu đồ mong muốn.

- **Node:** Các thực thể trong cơ sở dữ liệu đồ thị Neo4j được gọi là node. Các node được tham chiếu trong Cypher bằng dấu ngoặc đơn.

```
MATCH (n:Person {name:'John'})  
RETURN n.name AS Name
```

Trong ví dụ trên, (n:Person name:'John') là một node, node này bao gồm:

- Một nhãn (label) tên Person. Label giống như tag và được sử dụng để truy vấn cơ sở dữ liệu về các nút cụ thể. Một node có thể có nhiều label, ví dụ Person và Actor.
- Thuộc tính name có nội dung là John. Các thuộc tính được xác định trong dấu ngoặc nhọn, và được dùng để truy vấn thông tin cụ thể (như trong ví dụ trên là tìm tất cả các node có nhãn Person và thuộc tính name là John)
- Một biến n, các biến cho phép tham chiếu các node được chỉ định trong các mệnh đề tiếp theo. Trong ví dụ này, mệnh đề MATCH trả về kết quả gì thì kết quả đó sẽ được gán cho biến n. Biến n sau đó được chuyển đến mệnh đề RETURN tiếp theo. Mệnh đề RETURN in ra màn hình giá trị của một thuộc tính name của các node được lưu trong biến n.

- **Relationships:** Trong một đồ thị, các node có thể được kết nối với nhau bằng quan hệ (relationships). Một quan hệ phải có node bắt đầu, node kết thúc, và chính xác MỘT kiểu quan hệ. Trong Cypher, các quan hệ được biểu diễn bằng mũi tên chỉ định hướng của quan hệ.

```
MATCH (:Person {name: 'John'})-[r:KNOWS]->(friend:Person)
RETURN count(r) AS numberOfFriends
```

Trong ví dụ trên, relationship kiểu KNOWS được tìm kiếm. Truy vấn yêu cầu các quan hệ này xuất phát từ một node Person có tên là "John" đến bất kỳ nút Person nào khác. Hàm count() trong câu lệnh RETURN dùng để đếm tất cả các quan hệ được ràng buộc bởi biến r (nghĩa là đếm số người bạn mà John quen biết). Lưu ý:

- Trong khi các nút có thể có nhiều nhãn (labels), các relationship chỉ có thể có một kiểu (type) duy nhất.
  - Thông tin bên trong mẫu quan hệ phải được bao quanh bởi dấu ngoặc vuông.
- **Paths:** Đường dẫn (paths) trong một đồ thị bao gồm các node và relationship được kết nối với nhau. Việc khám phá các path là cốt lõi của ngôn ngữ Cypher.

```
MATCH (n:Person {name: 'John'})-[:KNOWS]-{1,5}(friend:Person
WHERE n.born < friend.born)
RETURN DISTINCT friend.name AS olderConnections
```

Truy vấn trên sử dụng một quan hệ định lượng để tìm tất cả các đường dẫn có tối đa 5 bước nhảy, chỉ đi qua các quan hệ kiểu KNOWS, bắt đầu từ nút Person tên John đến các nút Person khác có tuổi lớn hơn (theo điều kiện trong WHERE). Toán tử DISTINCT đảm bảo rằng câu lệnh RETURN chỉ trả về các nút duy nhất.

### 2.2.3 Một vài truy vấn cơ bản

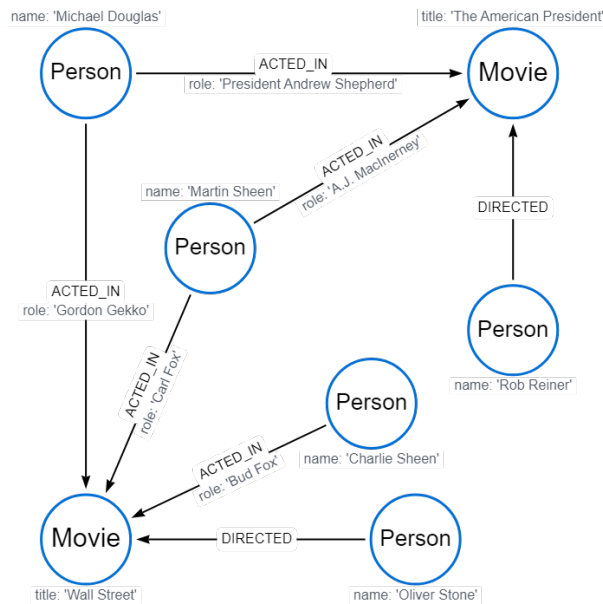
#### Tạo cơ sở dữ liệu

- Mệnh đề CREATE được sử dụng để tạo các node và relationship. Mệnh đề CREATE cho phép tạo một hoặc nhiều nút. Mỗi nút có thể được gán nhãn (labels) và thuộc tính (properties). Ta có thể gán mỗi nút vào một biến để tham chiếu trong các truy vấn sau. Nếu có nhiều nhãn, chúng được phân tách bằng dấu : hoặc &

```
CREATE (Micheal:Person:Actor {name: 'Micheal '}),
(oliver:Person:Director {name: 'Oliver Stone'})
```

- Quan hệ cũng có thể được tạo bằng mệnh đề CREATE. Không giống như nút, quan hệ luôn cần một kiểu quan hệ duy nhất và hướng. Tương tự như nút, các quan hệ có thể được gán thuộc tính và kiểu quan hệ, đồng thời được gán vào biến.

```
CREATE (charlie:Person:Actor
{name: 'Charlie Sheen'})-[:ACTED_IN {role: 'Bud Fox'}]
->(wallStreet:Movie {title: 'Wall Street'})
<-[:DIRECTED]-(oliver:Person:Director {name: 'Oliver Stone'})
```



Hình 2.2: Đồ thị được sử dụng cho các ví dụ trong mục 2.2.3

### Tìm kiếm node

Mệnh đề MATCH được sử dụng để tìm kiếm một mẫu cụ thể trong đồ thị, chẳng hạn như một nút cụ thể. Mệnh đề RETURN xác định phần nào của mẫu đồ thị tìm được sẽ được in ra.

- Truy vấn tìm tất cả các nút

```
MATCH (n) RETURN n
```

- Truy vấn tìm một nút nhất định:

```
MATCH (keanu:Person {name:'Keanu Reeves'})
RETURN keanu.name AS name, keanu.born AS born
```

- Ta có thể áp dụng các biểu thức logic trong truy vấn tìm kiếm:

```
MATCH (n:Movie|Person) RETURN n.name AS name, n.title AS title
```

### Tìm kiếm quan hệ

Mệnh đề MATCH cho phép bạn chỉ định các mẫu quan hệ có độ phức tạp khác nhau để truy vấn đồ thị. Không giống như mẫu nút, mẫu quan hệ phải có nút ở cả hai đầu.

- Mẫu quan hệ trống: Sử dụng - - để tìm các nút được kết nối mà không cần chỉ định kiểu quan hệ hay thuộc tính:

```
MATCH (:Person {name: 'Oliver Stone'})--(n)
RETURN n AS connectedNodes
```

- Mẫu quan hệ có hướng: Hướng của quan hệ được thể hiện qua mũi tên

```
MATCH (:Person {name: 'Oliver Stone'})-->(movie:Movie)
RETURN movie.title AS movieTitle
```

## Chương 3

# Phân tích dữ liệu hội thoại bằng Neo4j

### 3.1 Tổng quan bộ dữ liệu

Bộ dữ liệu **Chat Data** được thiết kế nhằm mô phỏng các tương tác thực tế giữa người dùng trong các phiên trò chuyện nhóm. Bộ dữ liệu tập trung vào việc ghi nhận các hành vi như tạo, tham gia, rời khỏi các phiên trò chuyện cũng như các tương tác giữa người dùng qua tin nhắn. Đây là một nguồn dữ liệu quan trọng để phân tích hành vi người dùng và mối quan hệ trong mạng lưới tương tác.

#### 3.1.1 Cấu trúc bộ dữ liệu

Bộ dữ liệu bao gồm 6 tệp CSV, mỗi tệp đại diện cho một khía cạnh cụ thể của các tương tác trong hệ thống:

Tệp CSV	Mục đích	Cột dữ liệu
chat_create_team_chat.csv	Ghi nhận các phiên trò chuyện được khởi tạo	userID, teamID, teamChatSessionID, timestamp
chat_join_team_chat.csv	Ghi nhận thông tin người dùng tham gia phiên trò chuyện	userID, teamChatSessionID, timestamp
chat_leave_team_chat.csv	Theo dõi thông tin người dùng rời khỏi phiên trò chuyện	userID, teamChatSessionID, timestamp
chat_item_team_chat.csv	Lưu lại các tin nhắn được gửi trong phiên trò chuyện	userID, teamChatSessionID, chatItemID, timestamp
chat_mention_team_chat.csv	Theo dõi việc người dùng được đề cập trong tin nhắn	chatItemID, userID, timestamp
chat_response_team_chat.csv	Ghi nhận các phản hồi giữa các tin nhắn	chatItemID_1, chatItemID_2, timestamp

#### 3.1.2 Đặc điểm nổi bật

- **Tính kết nối cao:** Bộ dữ liệu tập trung vào các mối quan hệ giữa người dùng, nhóm, phiên trò chuyện và tin nhắn. Các quan hệ như **"Creates"**, **"Joins"**, **"ResponseTo"** giúp dễ dàng mô hình hóa dữ liệu dưới dạng đồ thị.
- **Dữ liệu thời gian thực:** Mỗi sự kiện trong dữ liệu đều gắn nhãn thời gian (timestamp), cho phép phân tích lịch sử và xu hướng theo thời gian.

- **Khả năng mở rộng:** Bộ dữ liệu có thể được mở rộng để bổ sung thông tin như nội dung tin nhắn, cảm xúc người dùng, hoặc loại tương tác để hỗ trợ các phân tích phức tạp hơn.

### 3.1.3 Ứng dụng tiềm năng

- **Phân tích hành vi người dùng:**
  - Xác định người dùng và nhóm hoạt động tích cực nhất.
  - Phát hiện các nhóm có mức độ tương tác yếu để đề xuất giải pháp cải thiện.
- **Khám phá mạng lưới tương tác:**
  - Tìm chuỗi hội thoại dài nhất, qua đó đánh giá khả năng duy trì tương tác trong nhóm.
  - Phân tích các cụm người dùng với mức độ tương tác cao.
- **Tăng cường hỗ trợ ra quyết định:**
  - Sử dụng kết quả phân tích để tối ưu hóa công cụ trò chuyện, nâng cao hiệu quả giao tiếp trong nhóm.

## 3.2 Mô hình hóa bộ dữ liệu

### 3.2.1 Mô hình hóa cấu trúc đồ thị

Bộ dữ liệu **Chat Data** được mô hình hóa dưới dạng đồ thị để tận dụng sức mạnh xử lý mối quan hệ của Neo4j. Các thành phần chính bao gồm:

#### 1. Nút (Nodes):

- **User:** Đại diện cho người dùng
- **TeamChatSession:** Đại diện cho phiên trò chuyện nhóm.
- **ChatItem:** Đại diện cho tin nhắn.
- **Team:** Đại diện cho nhóm.

#### 2. Cạnh (Edges):

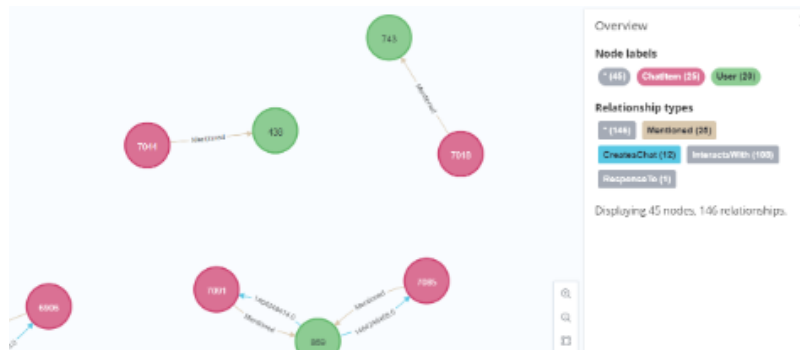
- **CreatesSession:** Người dùng tạo phiên trò chuyện.
- **Joins:** Người dùng tham gia phiên trò chuyện.
- **Leaves:** Người dùng rời phiên trò chuyện.
- **CreatesChat:** Người dùng gửi tin nhắn.
- **OwnedBy:** Phiên trò chuyện thuộc về nhóm.
- **PartOf:** Tin nhắn thuộc về một phiên trò chuyện.
- **Mentioned:** Tin nhắn đề cập đến người dùng.
- **ResponseTo:** Tin nhắn phản hồi một tin nhắn khác.

### 3.2.2 Biểu diễn mối quan hệ trong đồ thị

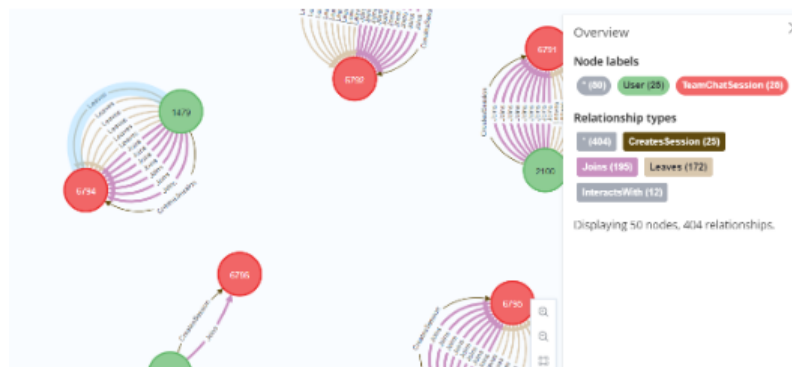
Cấu trúc đồ thị được mô tả như sau:

```
(User)-[:CreatesSession]->(TeamChatSession)-[:OwnedBy]->(Team)
(User)-[:Joins]->(TeamChatSession)-[:PartOf]->(ChatItem)
(ChatItem)-[:ResponseTo]->(ChatItem)
(ChatItem)-[:Mentioned]->(User)
```

Dưới đây là một số hình minh họa cho cơ sở dữ liệu, được lấy từ giao diện tương tác của Neo4j



Hình 3.1: Cấu trúc đồ thị



Hình 3.2: Người dùng tạo và tham gia phiên trò chuyện



Hình 3.3: Tin nhắn trong phiên trò chuyện

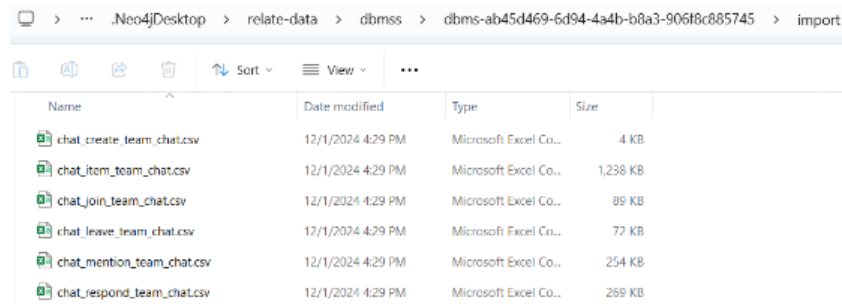


### 3.2.3 Tiến hành nhập dữ liệu

Để triển khai mô hình đồ thị, dữ liệu được nhập từ các tệp CSV qua các bước sau:

#### Chuẩn bị tập csv

Đặt các tệp CSV vào thư mục import của Neo4j:



Hình 3.4: Thêm các file dữ liệu .csv vào import của Neo4j

#### Nhập dữ liệu với Cypher

##### 1. Tạo nút (Node):

- Người dùng (User):

```
1 LOAD CSV WITH HEADERS FROM 'file:///chat_create_team_chat.csv' AS row
2 CREATE (:User {userID: toInteger(row.userID)});
```

- Phiên trò chuyện (TeamChatSession):

```
1 LOAD CSV WITH HEADERS FROM 'file:///chat_create_team_chat.csv' AS row
2 CREATE (:TeamChatSession {sessionID: toInteger(row.teamChatSessionID),
timestamp: row.timestamp});
```

- Tin nhắn (ChatItem):

```
1 LOAD CSV WITH HEADERS FROM 'file:///chat_item_team_chat.csv' AS row
2 CREATE (:ChatItem {chatItemID: toInteger(row.chatItemID), timestamp:
row.timestamp});
```

##### 2. Tạo cạnh (Edges):

- Người dùng tạo phiên trò chuyện:

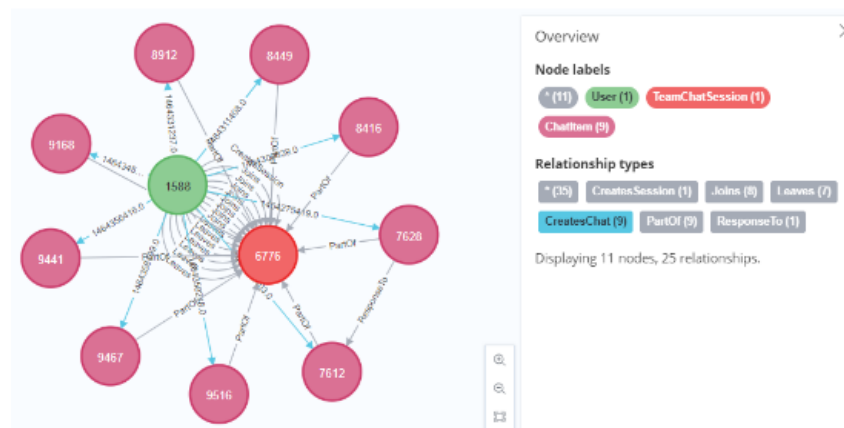
```
1 LOAD CSV WITH HEADERS FROM 'file:///chat_create_team_chat.csv' AS row
2 MATCH (u:User {userID: toInteger(row.userID)}), (s:TeamChatSession
{sessionID: toInteger(row.teamChatSessionID)})
3 CREATE (u)-[:CreatesSession {timestamp: row.timestamp}]->(s);
```

- Tin nhắn thuộc phiên trò chuyện:

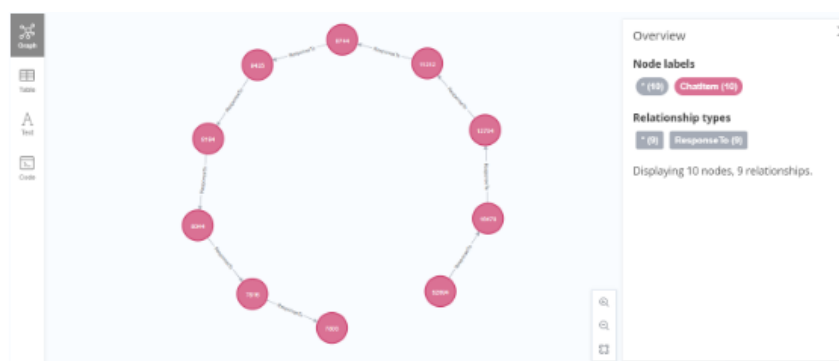
```
1 LOAD CSV WITH HEADERS FROM 'file:///chat_item_team_chat.csv' AS row
2 MATCH (c:ChatItem {chatItemID: toInteger(row.chatItemID)}),
(s:TeamChatSession {sessionID: toInteger(row.teamChatSessionID)})
3 CREATE (c)-[:PartOf]->(s);
```

### 3.3 Trực quan hóa và phân tích dữ liệu

### 3.3.1 Các truy vấn cơ bản



Hình 3.5: Quan hệ của một người dùng, phiên trò chuyện và tin nhắn



Hình 3.6: Chuỗi hội thoại dài nhất

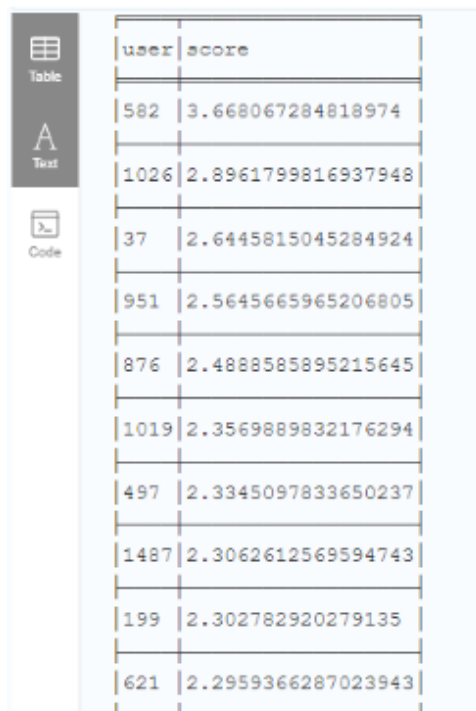
userId	chatCount	teamId	chatCount
194	115	82	1324
2067	111	195	1036
1087	109	112	957
209	109	18	844
554	107	194	836
1627	105	129	814
316	105	52	788
999	105	136	783
668	104	146	746
461	104	81	736

Hình 3.7: Trái: Các user tích cực nhất. Phải: Các team tích cực nhất

### 3.3.2 Phân tích nâng cao với Graph Data Science

1. **Xếp hạng người dùng dựa trên thuật toán PageRank:** Thuật toán PageRank được chạy trên đồ thị người dùng (userGraph) để đánh giá mức độ quan trọng của các người dùng trong mạng lưới dựa trên các mối quan hệ InteractsWith. Kết quả trả về bao gồm ID người dùng và điểm PageRank của họ, cụ thể:

- Top 3 người dùng quan trọng nhất:
  - User 582 với điểm PageRank cao nhất 3.6680.
  - User 1026 với điểm PageRank 2.8962.
  - User 37 với điểm PageRank 2.6446.
- Ý nghĩa:
  - Người dùng 582 là người quan trọng nhất trong mạng lưới, tức là họ nhận được nhiều kết nối có ảnh hưởng từ các người dùng khác.
  - Các người dùng tiếp theo như 1026 và 37 cũng đóng vai trò lớn trong việc duy trì sự kết nối của mạng.



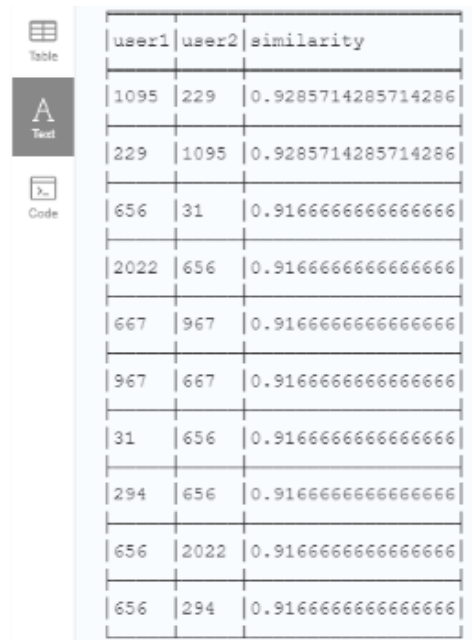
user	score
582	3.668067284818974
1026	2.8961799816937948
37	2.6445815045284924
951	2.5645665965206805
876	2.4888585895215645
1019	2.3569889832176294
497	2.3345097833650237
1487	2.3062612569594743
199	2.302782920279135
621	2.2959366287023943

Hình 3.8: Kết quả trả về sau khi dùng thuật toán PageRank

2. **Phân tích độ tương đồng giữa các người dùng bằng thuật toán Node Similarity:** Thuật toán NodeSimilarity cho kết quả bao gồm:

- Top 3 cặp người dùng tương đồng nhất:
  - User 1095 và User 229: Độ tương đồng 0.92857.
  - User 656 và User 31: Độ tương đồng 0.91667.
  - User 667 và User 967: Độ tương đồng 0.91667.
- Những người dùng có độ tương đồng cao thường có các mối quan hệ chung hoặc tham gia vào các hoạt động tương tự trong mạng lưới.

- Kết quả này hữu ích trong việc:
  - Gợi ý kết nối giữa các người dùng tương đồng.
  - Phân tích các nhóm người dùng có hành vi gần giống nhau.



user1	user2	similarity
1095	229	0.9285714285714286
229	1095	0.9285714285714286
656	31	0.9166666666666666
2022	656	0.9166666666666666
667	967	0.9166666666666666
967	667	0.9166666666666666
31	656	0.9166666666666666
294	656	0.9166666666666666
656	2022	0.9166666666666666
656	294	0.9166666666666666

Hình 3.9: Kết quả trả về sau khi dùng thuật toán NodeSimilarity

# Kết luận

Với kiến trúc linh hoạt cùng hiệu suất cao, Neo4j không chỉ dừng lại ở vai trò lưu trữ dữ liệu mà còn là một nền tảng mạnh mẽ cho các tác vụ liên quan đến khoa học dữ liệu và trí tuệ nhân tạo. Điểm nổi bật của Neo4j nằm ở khả năng xử lý các mối quan hệ phức tạp giữa các thực thể trong dữ liệu, giúp phân tích sâu các mô hình liên kết và trích xuất những thông tin giá trị từ mạng lưới dữ liệu phức tạp.

Thông qua bài tập lần này, nhóm chúng em đã có cơ hội được làm việc cùng với Neo4j nói riêng và dữ liệu dạng đồ thị nói chung, qua đó có thêm hiểu biết cũng như kinh nghiệm thực hiện xử lý và phân tích loại dữ liệu này. Trong tương lai, nhóm chúng em dự định cải tiến quy trình xử lý dữ liệu với Neo4j bằng cách tận dụng các nền tảng xử lý phân tán như Apache Spark. Sự kết hợp giữa sức mạnh đồ thị của Neo4j và khả năng tính toán song song của Spark mang lại hiệu suất vượt trội cùng khả năng mở rộng linh hoạt, đáp ứng nhu cầu của các ứng dụng dựa trên dữ liệu đồ thị ở quy mô lớn.

Như vậy, Neo4j không chỉ là một công cụ lưu trữ và xử lý dữ liệu đồ thị mạnh mẽ mà còn là một nền tảng toàn diện hỗ trợ phân tích dữ liệu, khám phá thông tin quan trọng và xây dựng các ứng dụng thông minh, mang lại giá trị to lớn trong kỷ nguyên dữ liệu hiện đại.

# Tài liệu tham khảo

- [1] Neo4j Documentation. <https://neo4j.com/docs/>.
- [2] Neo4j's Cypher Manual. <https://neo4j.com/docs/cypher/>.