**Tìm kiếm ảnh dựa trên đồ thị tri thức và đối tượng láng giềng**

**1. Xác định yêu cầu và mục tiêu**

* **Mục tiêu chính:** Mục tiêu của dự án này là tìm kiếm ảnh dựa trên mối quan hệ giữa các đối tượng trong ảnh. Thay vì chỉ tìm kiếm bằng từ khóa (ví dụ: "dog"), hệ thống sẽ dựa vào mối quan hệ giữa các đối tượng (ví dụ: "dog next to car") để tìm ảnh phù hợp. Điều này giúp nâng cao độ chính xác và tính thông minh trong tìm kiếm ảnh.
* **Ứng dụng thực tế:** Hệ thống này có thể hỗ trợ tìm kiếm ảnh thông minh trong các kho dữ liệu lớn như COCO. Nó cũng có thể hỗ trợ các hệ thống gợi ý hình ảnh, giúp người dùng tìm thấy ảnh phù hợp với yêu cầu tìm kiếm có ngữ cảnh.

**2. Những việc đã làm trong tuần dưới đây**

[Tiền xử lý dữ liệu COCO 2](#_Toc191656024)

[Xây dựng đồ thị tri thức từ COCO 6](#_Toc191656025)

[Tìm hiểu xây dựng mô hình tìm kiếm dựa trên láng giềng 10](#_Toc191656026)

[Công nghệ sử dụng 13](#_Toc191656027)

[Các mối quan hệ 14](#_Toc191656028)

[Một số ảnh sau khi sử dụng detectron2 để nhận diện 17](#_Toc191656029)

[Một vài truy vấn trong Neo4j sử dụng Cypher Query Language 22](#_Toc191656030)

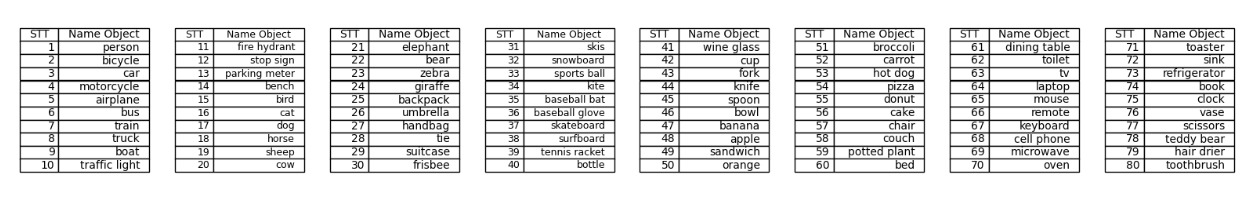
[Công việc tuần tiếp theo 25](#_Toc191656031)

# Tiền xử lý dữ liệu COCO

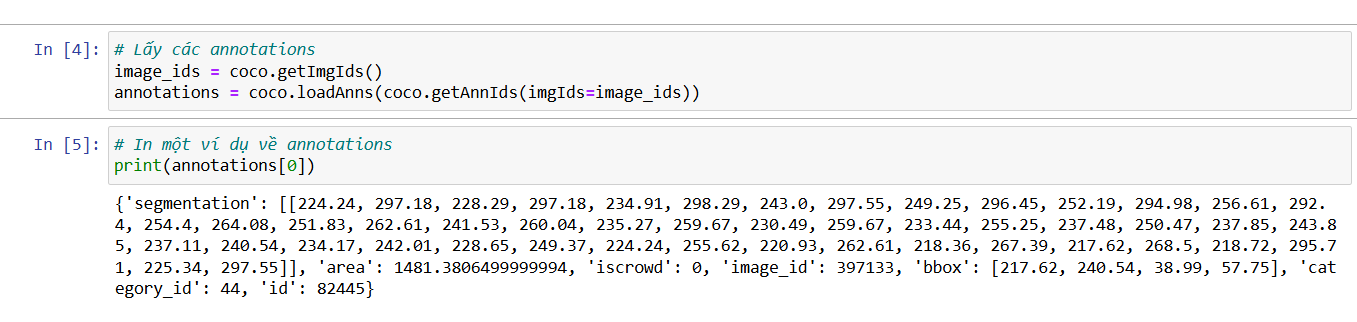
COCO (Common Objects in Context) là một bộ dữ liệu lớn gồm các ảnh và chú thích chi tiết, bao gồm các đối tượng và các mối quan hệ giữa chúng.

* **Dữ liệu đầu vào:**
* Các chú thích ảnh (captions): Mô tả ngắn gọn về các cảnh trong ảnh.
* Các đối tượng (categories): COCO bao gồm 80 danh mục đối tượng, ví dụ như "dog", "car", "tree".





* **Bounding box:** Vị trí của các đối tượng trong ảnh.



**Trong đó:**

1. **segmentation:** là dữ liệu phân đoạn (segmentation), mô tả hình dạng của đối tượng trong ảnh dưới dạng một danh sách các điểm tọa độ.

Các tọa độ này tạo thành một đa giác (polygon) bao quanh đối tượng.

Trong trường hợp này, các giá trị [224.24, 297.18, ...] là các tọa độ của các điểm tạo thành hình dạng của đối tượng.

1. **area:** là diện tích của đối tượng trong ảnh, tính theo số pixel.

Trong trường hợp này, area là diện tích của đối tượng trong ảnh là khoảng 1481.38 pixel.

1. **iscrowd:** là một cờ chỉ ra nếu đối tượng này là crowd (tập hợp các đối tượng không rõ ràng, ví dụ: nhóm người hoặc vật thể lộn xộn).

iscrowd = 0 có nghĩa là đối tượng này không phải là "crowd", tức là đây là một đối tượng đơn lẻ và dễ dàng nhận diện.

1. **image\_id:** là id của ảnh chứa đối tượng này.

image\_id = 397133 có nghĩa là đối tượng này xuất hiện trong ảnh có ID là 397133.

1. **bbox:** là bounding box (hộp giới hạn) của đối tượng trong ảnh, giúp xác định vị trí của đối tượng trong ảnh.

bbox = [217.62, 240.54, 38.99, 57.75] là tọa độ của hộp giới hạn:

217.62 là tọa độ x của góc trên bên trái.

240.54 là tọa độ y của góc trên bên trái.

38.99 là chiều rộng (width) của bounding box.

57.75 là chiều cao (height) của bounding box.

Hộp giới hạn giúp xác định vùng chứa đối tượng trong ảnh.

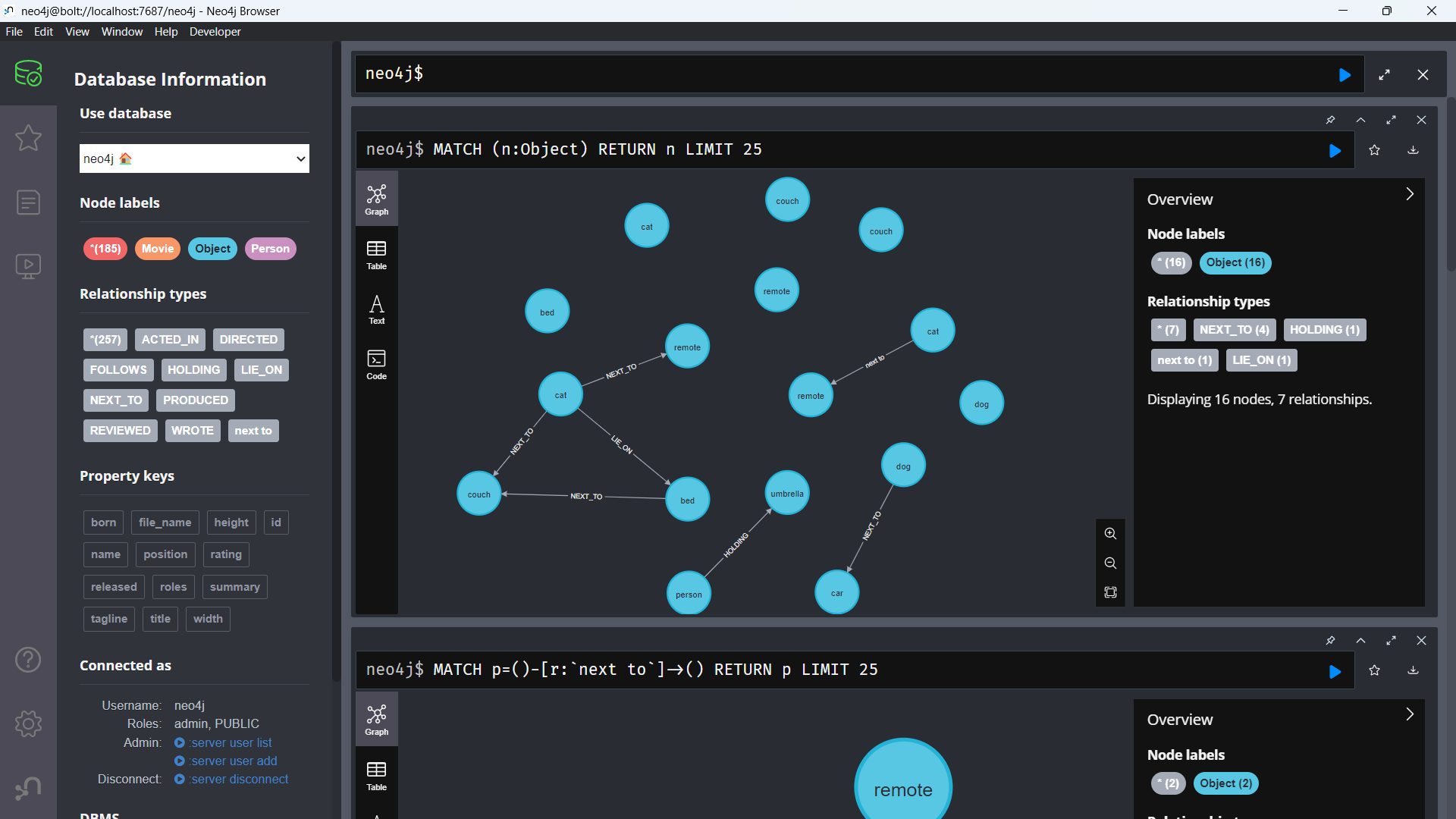
1. **category\_id:** Đây là ID của danh mục đối tượng trong COCO, chỉ ra loại đối tượng mà chú thích này thuộc về.

category\_id = 44 có nghĩa là đối tượng này thuộc loại "cat" (mèo), bởi vì trong COCO, ID 44 tương ứng với "cat" trong danh mục đối tượng.

1. **id:** là ID của đối tượng chú thích trong bộ dữ liệu.

id = 82445 là ID duy nhất của đối tượng này trong bộ dữ liệu COCO.

* **Quan hệ giữa các đối tượng:** Ví dụ: "dog next to car", "person holding umbrella".
* **Xây dựng đồ thị tri thức (KG):** Đồ thị tri thức (Knowledge Graph) sẽ có các nút đại diện cho các đối tượng và các cạnh đại diện cho các mối quan hệ giữa các đối tượng.
* **Nút (Nodes):** Các đối tượng trong ảnh (ví dụ: "dog", "car").
* **Cạnh (Edges):** Mối quan hệ giữa các đối tượng (ví dụ: "dog next to car").
* **Thuộc tính:** Vị trí, kích thước và số lượng đối tượng trong ảnh.

****

# Xây dựng đồ thị tri thức từ COCO

* **Trích xuất đối tượng**: Sử dụng mô hình nhận diện đối tượng như **Detectron2** để phát hiện các đối tượng trong ảnh. Sau đó, có thể lấy được danh sách các đối tượng và mối quan hệ giữa chúng.



**Trong đó:**

* cfg = get\_cfg(): Khởi tạo một đối tượng cấu hình (cfg) cho mô hình. get\_cfg() là một hàm của Detectron2 để tạo một cấu hình mặc định cho mô hình.
* cfg.merge\_from\_file(): Hàm này tải và áp dụng cấu hình từ một tệp YAML có sẵn. Trong trường hợp này, nó lấy tệp cấu hình faster\_rcnn\_R\_50\_FPN\_3x.yaml từ model\_zoo của Detectron2. Đây là cấu hình cho mô hình Faster R-CNN với ResNet-50 và FPN (Feature Pyramid Networks), sử dụng cho bài toán nhận diện đối tượng trên bộ dữ liệu COCO.
* cfg.MODEL.WEIGHTS: Cấu hình trọng số (weights) của mô hình. Đoạn mã trên lấy trọng số được huấn luyện sẵn từ model\_zoo của Detectron2 cho mô hình Faster R-CNN với cấu hình ResNet-50 và FPN. Trọng số này đã được huấn luyện trên bộ dữ liệu COCO và có thể sử dụng để nhận diện đối tượng.
* cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5: Thiết lập ngưỡng (threshold) cho các đối tượng được nhận diện. Nếu xác suất (score) của đối tượng thấp hơn 0.5, đối tượng đó sẽ bị loại bỏ. Mức ngưỡng này có thể được điều chỉnh tùy thuộc vào yêu cầu của bài toán. Thông thường, giá trị này được chọn để giảm bớt các dự đoán sai (false positives).
* cfg.MODEL.DEVICE = "cpu": Đây là cấu hình để mô hình chạy trên CPU thay vì GPU. Nếu bạn có GPU và muốn sử dụng GPU để tăng tốc quá trình nhận diện, bạn có thể thay cpu bằng cuda (ví dụ: cfg.MODEL.DEVICE = "cuda"). Lưu ý rằng việc sử dụng GPU yêu cầu bạn phải cài đặt CUDA đúng cách.
* predictor = DefaultPredictor(cfg): Tạo một đối tượng predictor sử dụng cấu hình cfg. DefaultPredictor là một lớp trong Detectron2 dùng để thực hiện việc nhận diện đối tượng. Khi bạn gọi phương thức predictor(image), mô hình sẽ trả về các kết quả nhận diện từ ảnh đầu vào.

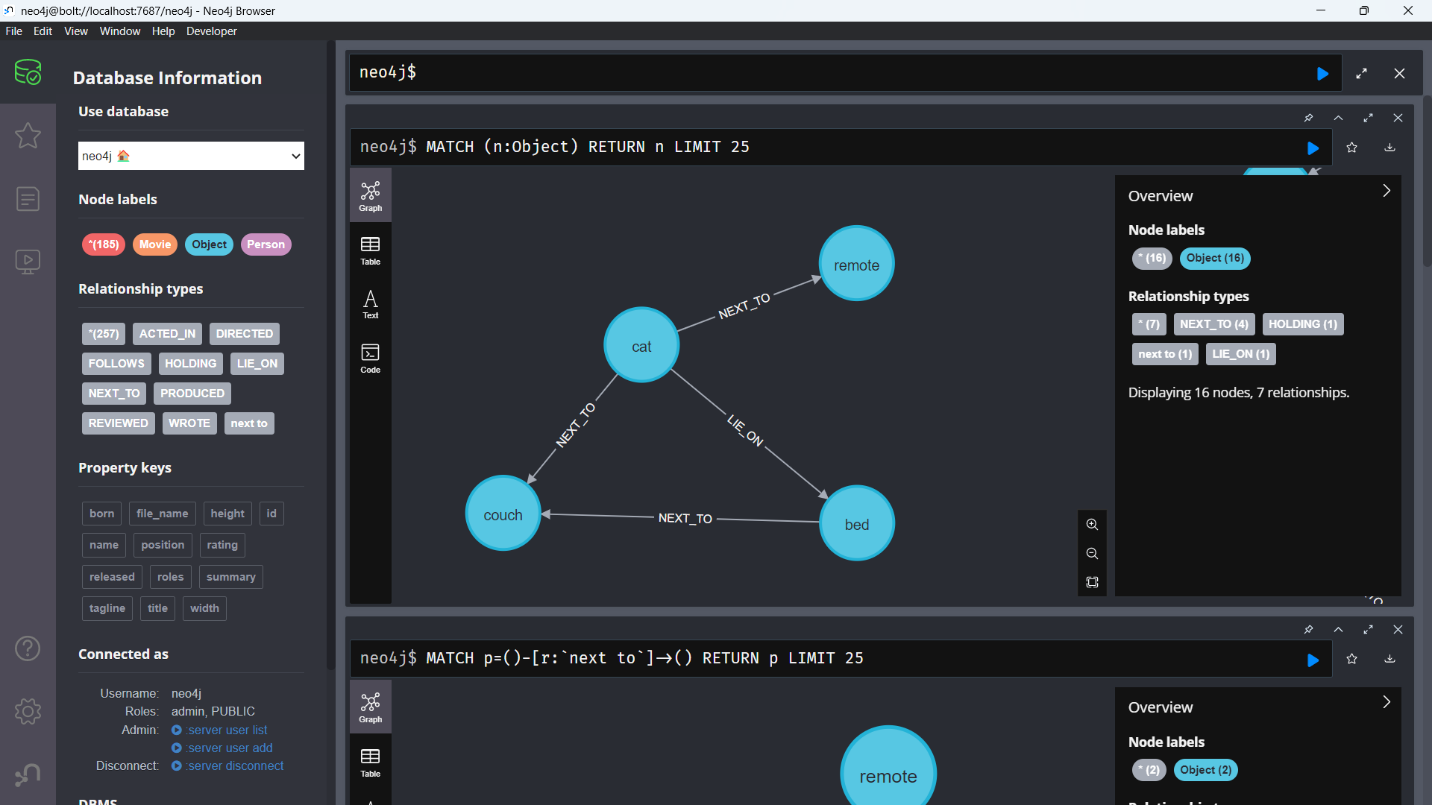
|  |  |
| --- | --- |
|  |  |
|  |  |

* **Biểu diễn đồ thị tri thức**: Dữ liệu sẽ được lưu dưới dạng **RDF triplets** hoặc **Knowledge Graph Embeddings**.
* Ví dụ RDF: (dog, next\_to, car), (person, holding, umbrella).

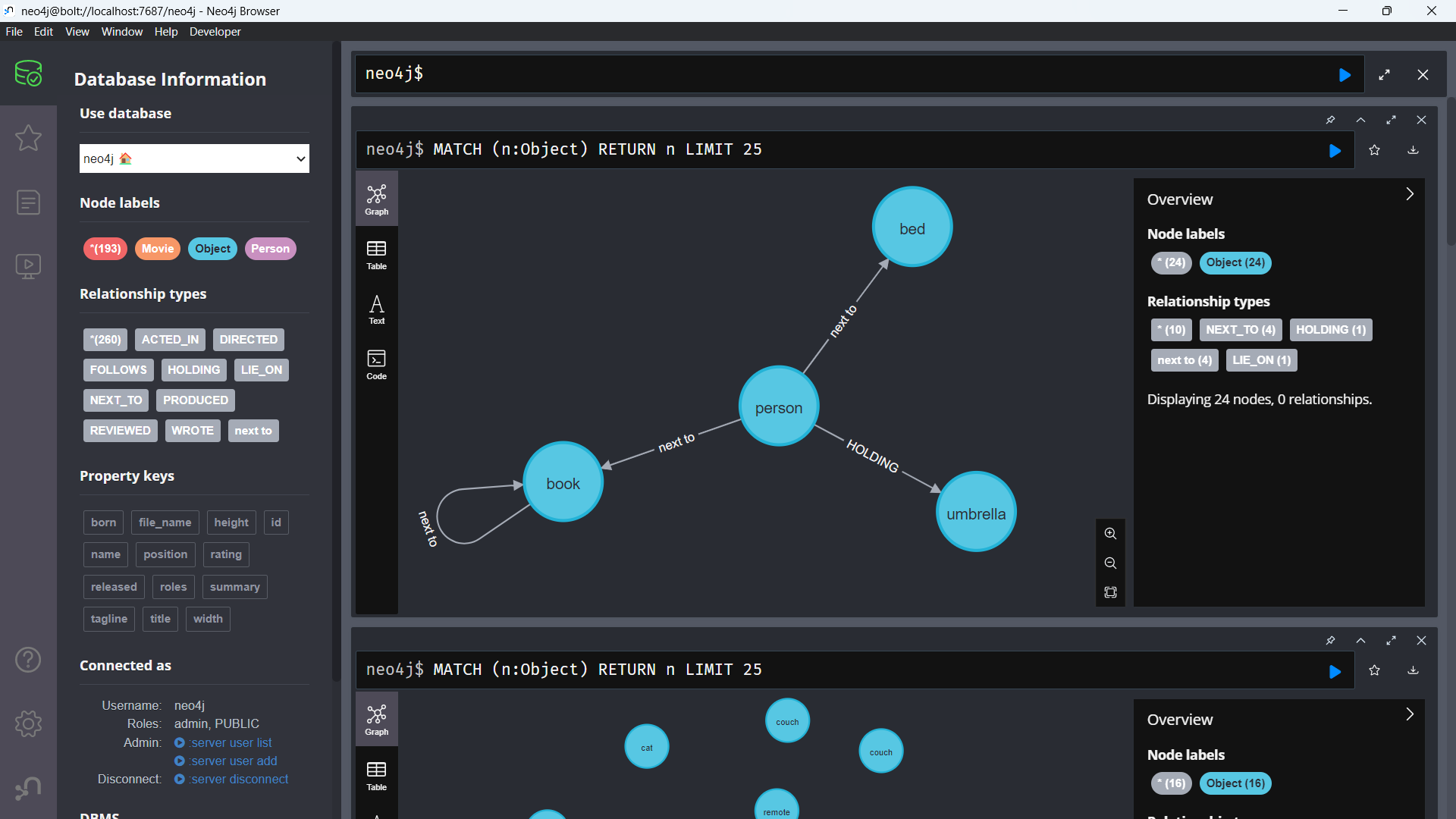


* **Lưu trữ KG**: Dữ liệu đồ thị tri thức sẽ được lưu trữ trong cơ sở dữ liệu đồ thị như Neo4j. Đây là nơi bạn sẽ tổ chức và lưu trữ tất cả các mối quan hệ giữa các đối tượng.





**Neo4j** là một cơ sở dữ liệu đồ thị, sử dụng ngôn ngữ truy vấn **Cypher** để thực hiện các thao tác trên đồ thị.



# Tìm hiểu xây dựng mô hình tìm kiếm dựa trên láng giềng

**Graph Neural Networks (GNN)**: là một loại mạng neural được thiết kế để làm việc với dữ liệu có cấu trúc đồ thị. Cấu trúc đồ thị bao gồm các nút (nodes) và cạnh (edges), và GNN giúp học các embedding (đại diện) cho các nút và cạnh trong đồ thị.

**Mục tiêu của GNN trong bài toán này:**

* Mô hình GNN sẽ học các **embedding** cho các đối tượng trong ảnh (ví dụ: "dog", "car", "tree") và các mối quan hệ giữa chúng (ví dụ: "dog next to car").
* Các e**mbedding** này giúp hệ thống nhận diện mối quan hệ gần gũi giữa các đối tượng và tìm kiếm ảnh dựa trên mối quan hệ giữa chúng.
* Các bước cơ bản trong GNN:
* **Input Graph**: Đầu vào là đồ thị tri thức (Knowledge Graph), nơi mỗi đối tượng là một nút và mỗi mối quan hệ giữa các đối tượng là một cạnh.
* **Message Passing**: GNN học thông qua message passing – mỗi nút nhận thông tin từ các nút láng giềng (neighbors) của nó thông qua các cạnh trong đồ thị.
* **Graph Convolution**: Sau khi thông tin được truyền qua các cạnh, các nút sử dụng các phép toán (ví dụ: convolution) để cập nhật các đại diện (embedding) của chúng.
* **Learning Embeddings**: Sau quá trình huấn luyện, mỗi nút trong đồ thị có một embedding đại diện cho thông tin về nó và các mối quan hệ của nó trong đồ thị.
* **HITS / PageRank**:

**PageRank:** PageRank là thuật toán được phát triển bởi Google để xác định tầm quan trọng của các trang web trong mạng internet. Thuật toán này sử dụng các liên kết (links) giữa các trang web để đánh giá mức độ quan trọng của chúng.

* **Ý tưởng:** Mỗi nút trong đồ thị sẽ nhận được một giá trị PageRank (điểm số), mà giá trị này phụ thuộc vào số lượng và chất lượng các nút khác liên kết đến nó.
* **Ứng dụng trong tìm kiếm ảnh:** Sử dụng PageRank để xác định các đối tượng quan trọng trong đồ thị tri thức, ví dụ: đối tượng nào có nhiều mối quan hệ với các đối tượng khác.

**HITS (Hyperlink-Induced Topic Search):** HITS là một thuật toán tương tự như PageRank nhưng có sự phân biệt rõ ràng hơn giữa hub và authorities.

* Hub là các nút chỉ dẫn đến các nút quan trọng khác (có nhiều liên kết ra ngoài).
* Authority là các nút có nhiều liên kết đến từ các hub quan trọng
* **Khai thác thông tin láng giềng**: Khi có một truy vấn như "dog next to car", bạn sẽ tìm các ảnh có đối tượng "dog" và "car" có mối quan hệ gần gũi. Sau đó, tính toán độ tương đồng giữa các ảnh này bằng **cosine similarity** trên các vector embedding.

**Các bước thực hiện:**

1. **Truy vấn đồ thị:**
   * Chuyển câu truy vấn "dog next to car" thành một **đồ thị con** (subgraph) trong đồ thị tri thức. Bạn có thể sử dụng các **công cụ** như **Cypher Query Language** trong Neo4j để tìm các mối quan hệ này.
2. **Tìm các ảnh có đối tượng "dog" và "car":**
   * Sử dụng truy vấn để tìm các ảnh chứa đối tượng "dog" và "car". Mỗi đối tượng sẽ được đại diện bởi một nút trong đồ thị, và mối quan hệ "next to" sẽ là cạnh giữa chúng.
3. **Tính toán độ tương đồng giữa các ảnh:**
   * **Cosine similarity** sẽ được sử dụng để đo độ tương đồng giữa các ảnh dựa trên các **vector embedding** mà bạn đã học được từ GNN. Bạn sẽ tính toán độ tương đồng giữa các vector embedding của các ảnh có "dog" và "car" để tìm ra những ảnh tương tự.

**Bước đầu tiếp cận:**

**Bước 1: Xây dựng đồ thị tri thức**

* Tạo các nút cho các đối tượng trong ảnh.
* Tạo các cạnh để biểu diễn mối quan hệ giữa các đối tượng (ví dụ: "dog next to car").
* Lưu trữ đồ thị tri thức này trong Neo4j.

**Bước 2: Sử dụng GNN để học các embedding cho các đối tượng**

* Xây dựng mô hình GNN với PyTorch Geometric hoặc DGL.
* Lưu các embedding học được cho các đối tượng trong đồ thị.

**Bước 3: Tính toán độ tương đồng giữa các ảnh**

* Khi nhận được một truy vấn như "dog next to car", sử dụng các embedding đã học để tính toán độ tương đồng giữa các ảnh có chứa đối tượng "dog" và "car".

**Bước 4: Triển khai mô hình tìm kiếm**

* Xây dựng một API tìm kiếm để nhận truy vấn từ người dùng, chuyển đổi chúng thành đồ thị con và tìm kiếm trong Neo4j.
* Xây dựng giao diện tìm kiếm để hiển thị kết quả.

**Công cụ và Thư viện**

* **PyTorch Geometric** hoặc **DGL**: Để xây dựng và huấn luyện GNN.
* **Neo4j**: Để lưu trữ và truy vấn đồ thị tri thức.
* **Cypher Query Language**: Để thực hiện các truy vấn trong Neo4j.
* **Scikit-learn**: Để tính toán cosine similarity giữa các embedding của ảnh.

# Công nghệ sử dụng

* **Xử lý ảnh**:
* **Detectron2**: Để phát hiện đối tượng trong ảnh.
* **OpenCV**: Thực hiện các thao tác xử lý ảnh cơ bản như thay đổi kích thước, cắt ảnh.
* **PyTorch**: Sử dụng cho việc huấn luyện các mô hình học sâu như GNN.
* **Đồ thị tri thức**:
* **Neo4j**: Là cơ sở dữ liệu đồ thị.
* **NetworkX**: Thư viện Python để làm việc với đồ thị.
* **DGL (Deep Graph Library)**: Thư viện hỗ trợ học sâu cho đồ thị.
* **Mô hình tìm kiếm ảnh**:
* **Graph Neural Networks** **(GNN)**.
* **Knowledge Graph Embeddings** như **TransE**, **TransH**.

# Các mối quan hệ

|  |  |
| --- | --- |
| **1** | "ON" (Một đối tượng nằm trên đối tượng khác): Nếu bounding box của đối tượng A nằm hoàn toàn trong phạm vi của đối tượng B (theo trục y), và trục x của A nằm trong phạm vi của B.  Điều kiện: y1(A) < y1(B) and y2(A) > y2(B) và x1(A) < x1(B) and x2(A) > x2(B).  Ví dụ: "The cat is on the table" (Con mèo nằm trên bàn). |
| **2** | "NEXT\_TO" (Một đối tượng nằm bên cạnh đối tượng khác): Hai đối tượng gần nhau nhưng không chồng lên nhau. Khoảng cách giữa trung tâm của hai đối tượng phải nhỏ hơn một ngưỡng nhất định.  Điều kiện: Khoảng cách giữa hai trung tâm (calculate\_distance(center(A), center(B))) < ngưỡng (ví dụ 100 pixel).  Ví dụ: "The remote is next to the cat" (Điều khiển nằm bên cạnh con mèo). |
| **3** | "NEAR" (Một đối tượng gần đối tượng khác): Hai đối tượng gần nhau, nhưng không đủ gần để là "NEXT\_TO". Khoảng cách giữa chúng vẫn nhỏ, nhưng lớn hơn ngưỡng của "NEXT\_TO".  Điều kiện: Khoảng cách giữa hai đối tượng (calculate\_distance(center(A), center(B))) < ngưỡng lớn hơn ngưỡng "NEXT\_TO" nhưng nhỏ hơn ngưỡng "FAR".  Ví dụ: "The chair is near the desk" (Cái ghế gần cái bàn). |
| **4** | "UNDER" (Một đối tượng nằm dưới đối tượng khác): Nếu bounding box của đối tượng A nằm dưới đối tượng B (trục y của A lớn hơn trục y của B).  Điều kiện: y1(A) > y2(B) và x1(A) < x2(B) và x2(A) > x1(B).  Ví dụ: "The cat is under the table" (Con mèo dưới bàn).  "ABOVE" (Một đối tượng nằm phía trên đối tượng khác): Nếu bounding box của đối tượng A nằm trên đối tượng B (trục y của A nhỏ hơn trục y của B).  Điều kiện: y2(A) < y1(B) và x1(A) < x2(B).  Ví dụ: "The lamp is above the desk" (Đèn ở trên bàn). |
| **5** | "INSIDE" (Một đối tượng nằm trong đối tượng khác): Nếu bounding box của đối tượng A hoàn toàn nằm trong bounding box của đối tượng B.  Điều kiện: x1(A) >= x1(B) and x2(A) <= x2(B) và y1(A) >= y1(B) and y2(A) <= y2(B).  Ví dụ: "The book is inside the box" (Cuốn sách trong hộp). |
| **6** | "BEHIND" (Một đối tượng nằm phía sau đối tượng khác): Nếu bounding box của đối tượng A nằm ở phía sau đối tượng B, tức là trục y của A lớn hơn trục y của B.  Điều kiện: y1(A) > y2(B) và x1(A) < x2(B).  Ví dụ: "The cat is behind the couch" (Con mèo ở phía sau ghế sofa). |
| **7** | "IN\_FRONT\_OF" (Một đối tượng nằm phía trước đối tượng khác): Nếu bounding box của đối tượng A nằm phía trước đối tượng B, tức là trục y của A nhỏ hơn trục y của B.  Điều kiện: y2(A) < y1(B) và x1(A) < x2(B).  Ví dụ: "The dog is in front of the door" (Con chó ở phía trước cửa). |
| **8** | "TO\_THE\_LEFT\_OF" (Một đối tượng nằm bên trái đối tượng khác): Nếu trục x của đối tượng A nhỏ hơn trục x của đối tượng B.  Điều kiện: x2(A) < x1(B).  Ví dụ: "The book is to the left of the lamp" (Cuốn sách nằm bên trái cái đèn). |
| **9** | "TO\_THE\_RIGHT\_OF" (Một đối tượng nằm bên phải đối tượng khác): Nếu trục x của đối tượng A lớn hơn trục x của đối tượng B.  Điều kiện: x1(A) > x2(B).  Ví dụ: "The pen is to the right of the book" (Cây bút nằm bên phải cuốn sách). |
| **10** | "CLOSE\_TO" (Một đối tượng gần đối tượng khác): Khoảng cách giữa các đối tượng nhỏ hơn một ngưỡng nhất định.  Điều kiện: calculate\_distance(center(A), center(B)) < threshold.  Ví dụ: "The sofa is close to the wall" (Cái sofa gần tường). |

# Hàm tính khoảng cách giữa hai đối tượng (center A, center B)

def calculate\_distance(center\_A, center\_B):

return math.sqrt((center\_A[0] - center\_B[0])\*\*2 + (center\_A[1] - center\_B[1])\*\*2)

# Hàm xác định quan hệ giữa hai đối tượng dựa trên bounding box

def determine\_relationship(obj1, obj2):

# Lấy center của mỗi object (trung tâm)

center\_A = ((obj1["x1"] + obj1["x2"]) / 2, (obj1["y1"] + obj1["y2"]) / 2)

center\_B = ((obj2["x1"] + obj2["x2"]) / 2, (obj2["y1"] + obj2["y2"]) / 2)

distance = calculate\_distance(center\_A, center\_B)

# Quan hệ "ON"

if obj1["y1"] < obj2["y1"] and obj1["y2"] > obj2["y2"] and obj1["x1"] < obj2["x1"] and obj1["x2"] > obj2["x2"]:

return "ON"

# Quan hệ "NEXT\_TO"

elif abs(center\_A[0] - center\_B[0]) < 50 and abs(center\_A[1] - center\_B[1]) < 50:

return "NEXT\_TO"

# Quan hệ "NEAR"

elif distance < 100:

return "NEAR"

# Quan hệ "UNDER"

elif obj1["y1"] > obj2["y2"] and obj1["x1"] < obj2["x2"] and obj1["x2"] > obj2["x1"]:

return "UNDER"

# Quan hệ "ABOVE"

elif obj1["y2"] < obj2["y1"] and obj1["x1"] < obj2["x2"]:

return "ABOVE"

# Quan hệ "INSIDE"

elif obj1["x1"] >= obj2["x1"] and obj1["x2"] <= obj2["x2"] and obj1["y1"] >= obj2["y1"] and obj1["y2"] <= obj2["y2"]:

return "INSIDE"

# Quan hệ "BEHIND"

elif obj1["y1"] > obj2["y2"] and obj1["x1"] < obj2["x2"]:

return "BEHIND"

# Quan hệ "IN\_FRONT\_OF"

elif obj1["y2"] < obj2["y1"] and obj1["x1"] < obj2["x2"]:

return "IN\_FRONT\_OF"

# Quan hệ "TO\_THE\_LEFT\_OF"

elif obj1["x2"] < obj2["x1"]:

return "TO\_THE\_LEFT\_OF"

# Quan hệ "TO\_THE\_RIGHT\_OF"

elif obj1["x1"] > obj2["x2"]:

return "TO\_THE\_RIGHT\_OF"

# Quan hệ "CLOSE\_TO"

elif distance < 100:

return "CLOSE\_TO"

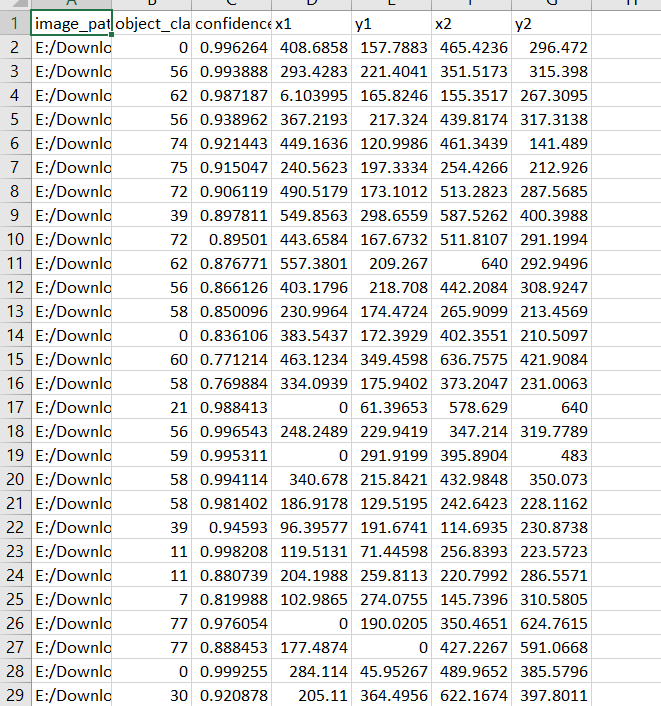
return None

# Một số ảnh sau khi sử dụng detectron2 để nhận diện

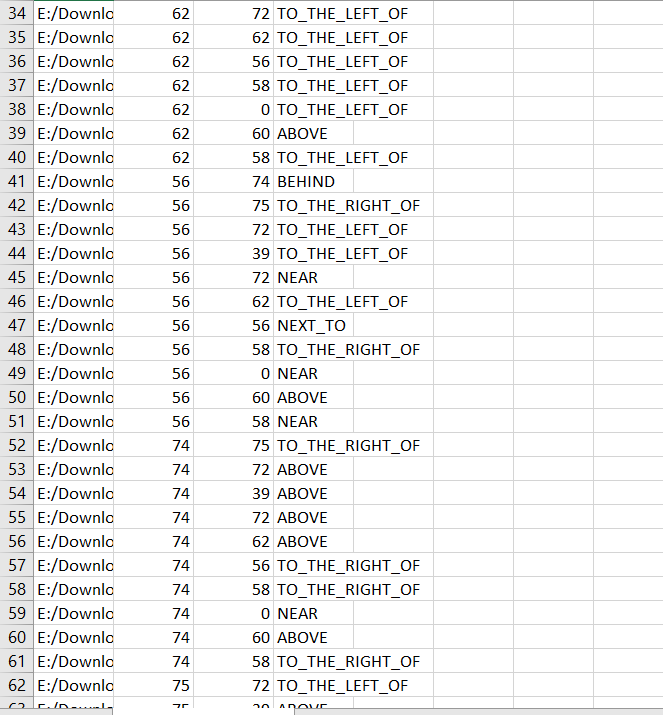
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Sau khi nhận dạng thì thêm vào file .csv và chuyển sang lưu trữ trong cơ sở dữ liệu Neo4j**

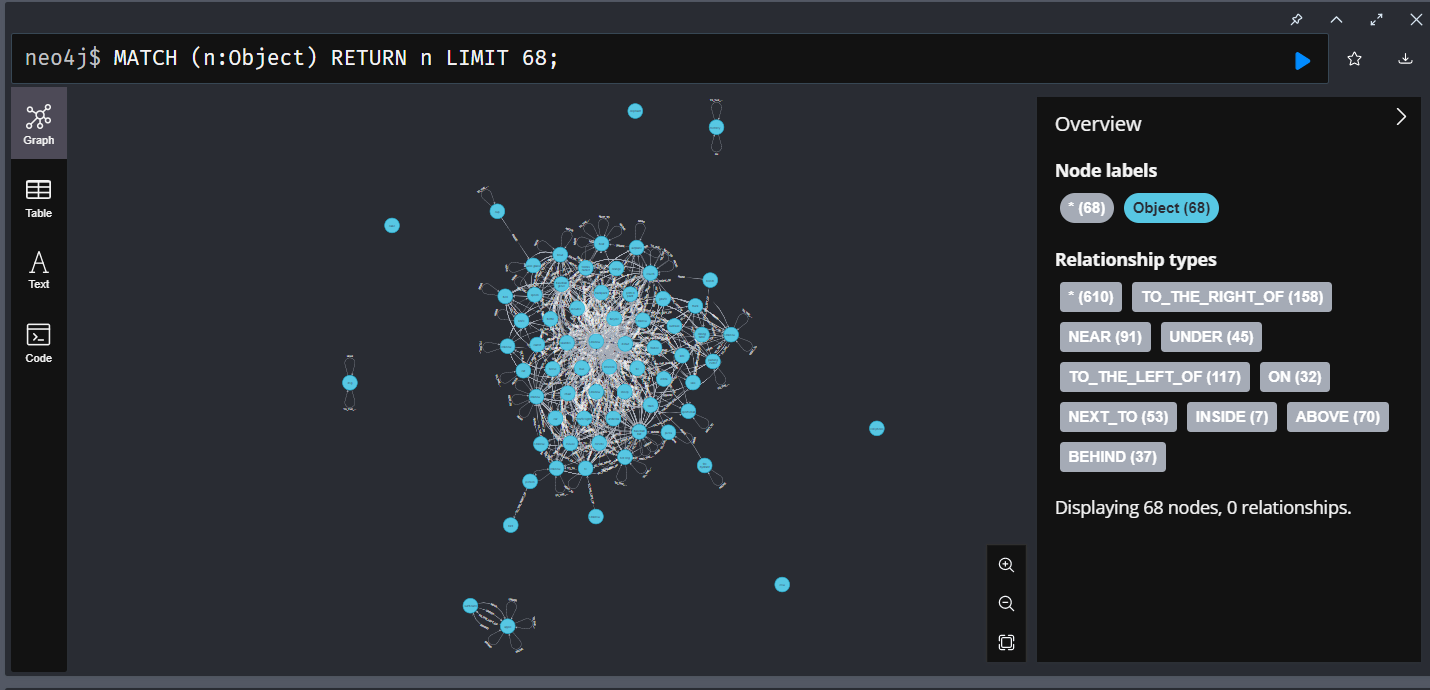
* **Bao gồm 576 objects đã nhận diện được trên 100 images**

****

* **Bao gồm 2705 relationship trên 567 objects**

****

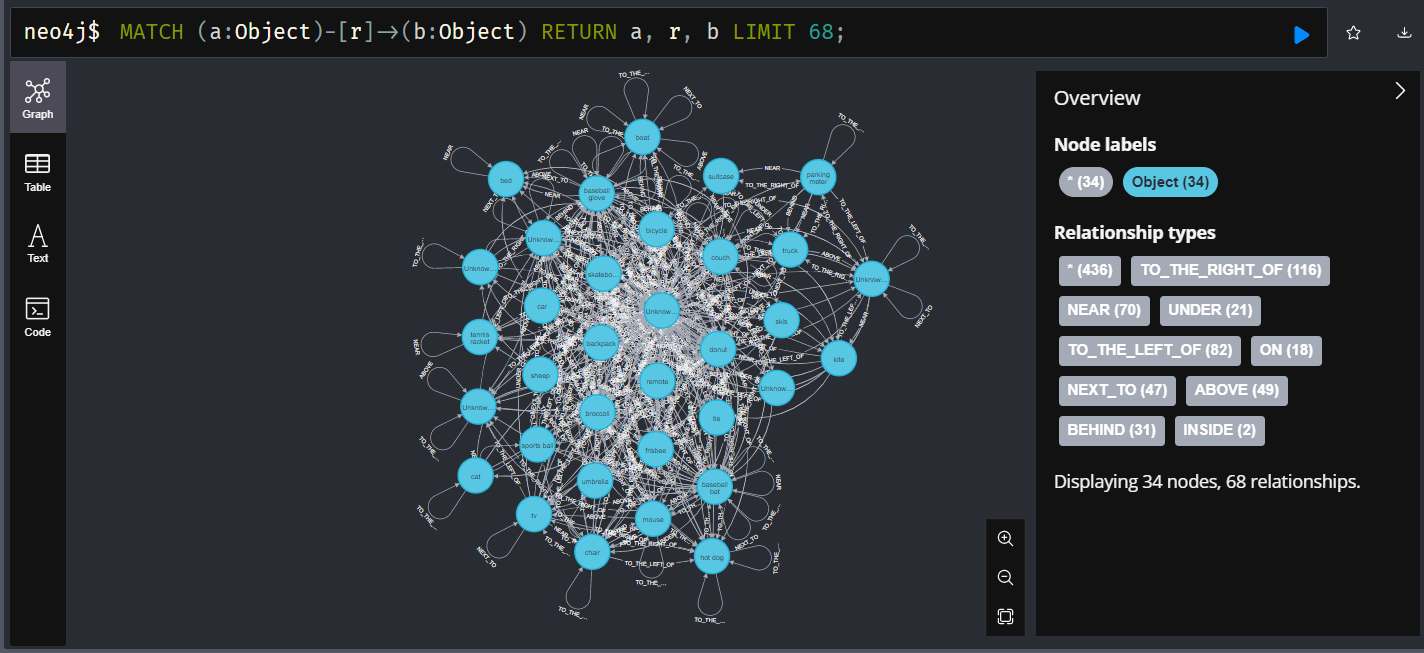
* **Lưu trữ vào cơ sở dữ liệu Neo4j**

****

# Một vài truy vấn trong Neo4j sử dụng Cypher Query Language

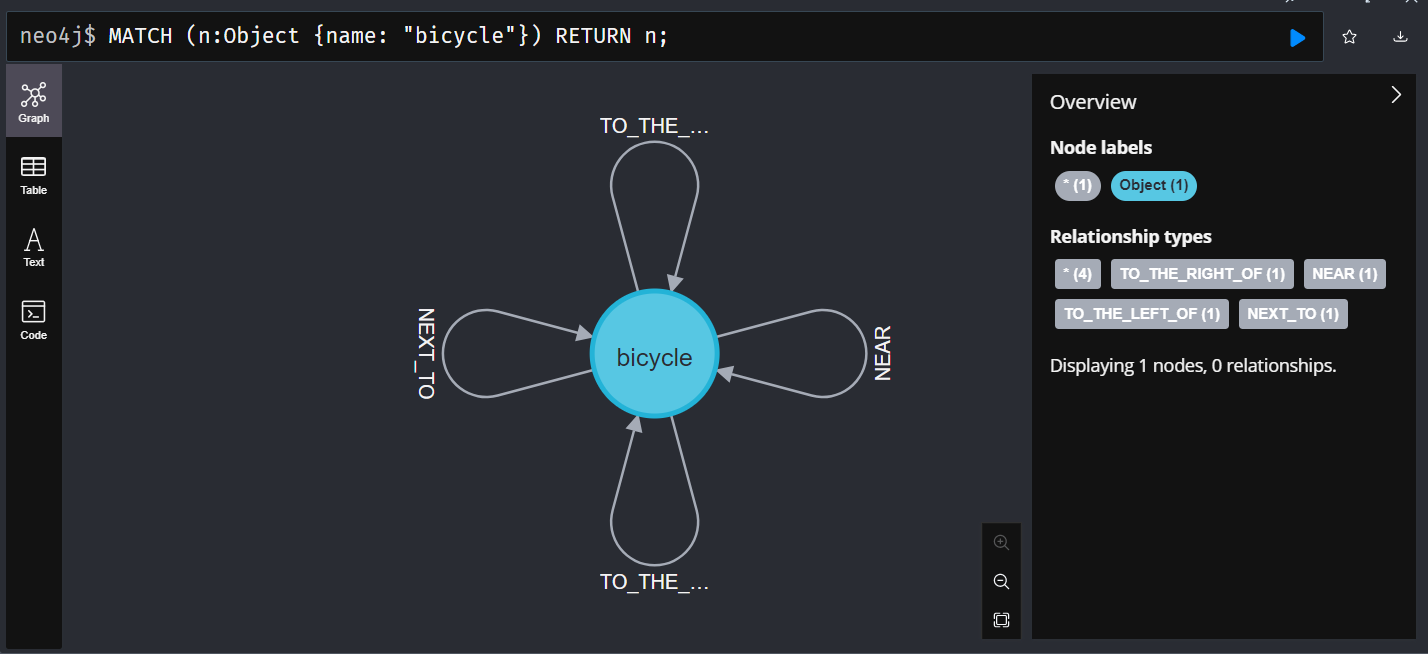
MATCH (a:Object)-[r]->(b:Object) RETURN a, r, b LIMIT 68;

Truy vấn này sẽ lấy tất cả các **relationship** giữa các node kiểu **"Object"** và hiển thị các node liên quan.

****

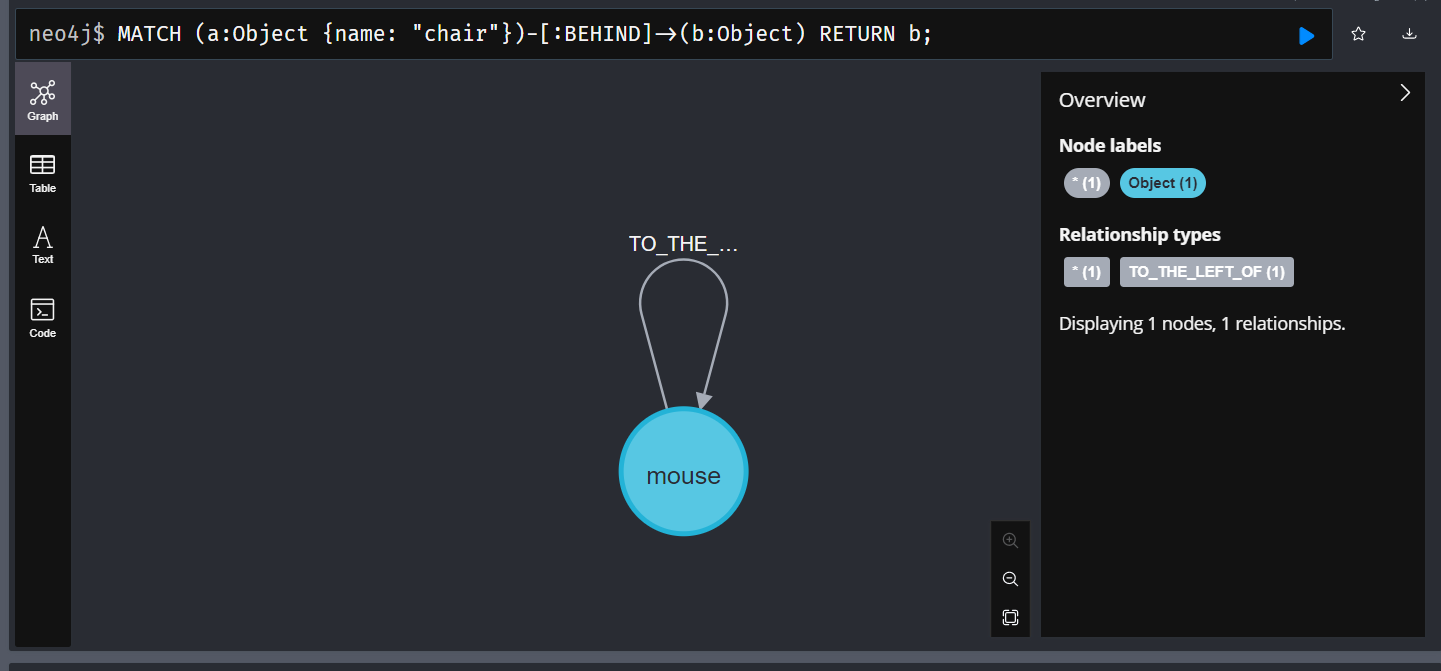
MATCH (n:Object {name: "bicycle"}) RETURN n;

Truy vấn này sẽ tìm các node có **tên là "bicycle"** trong thuộc tính name.



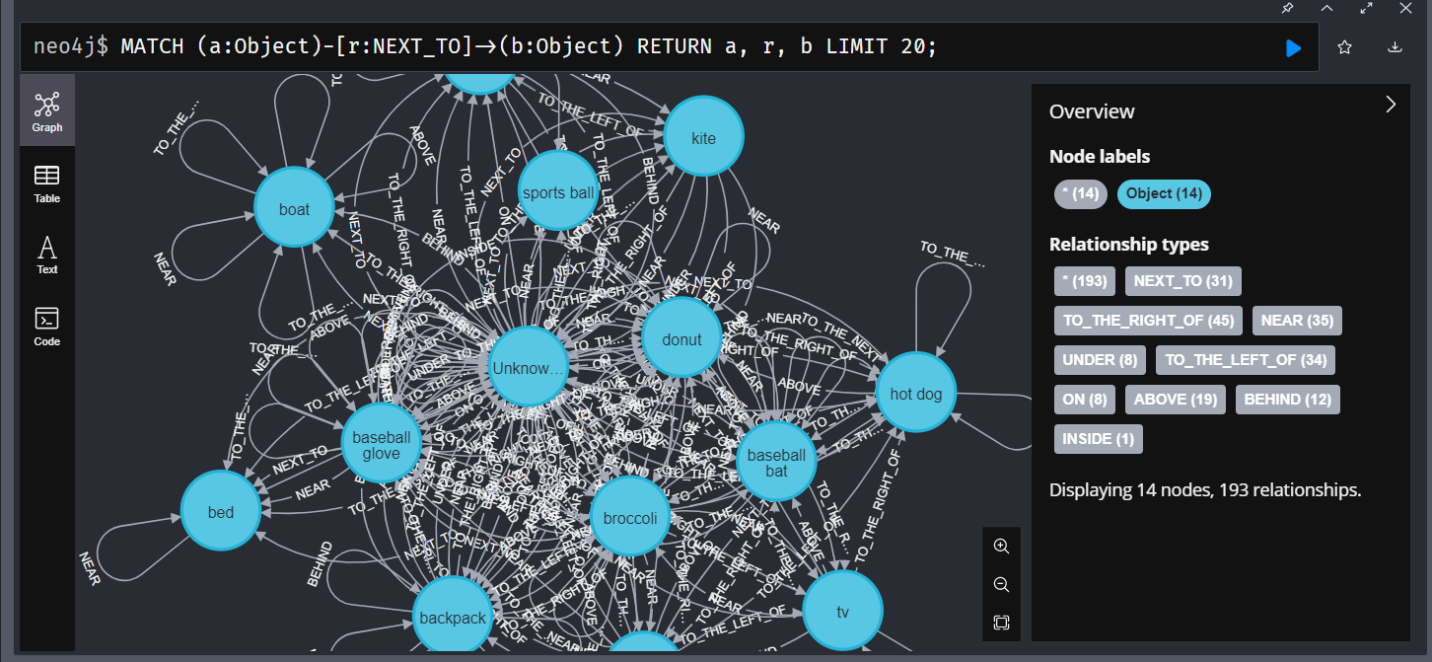
MATCH (a:Object {name: "chair"})-[:BEHIND]->(b:Object) RETURN b;

Truy vấn này sẽ tìm tất cả các node "Object" có quan hệ BEHINDvới node có tên **"chair"**.



MATCH (a:Object)-[r:NEXT\_TO]->(b:Object) RETURN a, r, b LIMIT 20;

Truy vấn này sẽ tìm tất cả các quan hệ **"NEXT\_TO"** giữa các đối tượng.



# Công việc tuần tiếp theo

**1. Cải thiện phần Lưu trữ KG (Neo4j):** Tiếp tục tối ưu và hoàn thiện việc lưu trữ Knowledge Graph (KG) trong Neo4j, đảm bảo rằng tất cả các đối tượng và mối quan hệ đã được lưu trữ chính xác.

* **Kiểm tra và tối ưu hóa việc lưu trữ**: Đảm bảo rằng các đối tượng (nút) và mối quan hệ (cạnh) trong Neo4j được lưu trữ hiệu quả. Bạn có thể kiểm tra dữ liệu trong Neo4j và tối ưu các chỉ mục (index) cho các trường như category\_id, image\_id để tăng tốc truy vấn.
* **Tạo các truy vấn trong Neo4j**: Viết các truy vấn Cypher để lấy dữ liệu từ KG, ví dụ: tìm tất cả các đối tượng có mối quan hệ với "dog" hoặc "car".
* **Kiểm tra tính chính xác của dữ liệu**: Đảm bảo rằng tất cả các đối tượng và mối quan hệ trong ảnh COCO đã được trích xuất chính xác và lưu vào Neo4j.

**2. Xây dựng mô hình tìm kiếm dựa trên láng giềng:** Triển khai mô hình tìm kiếm ảnh dựa trên mối quan hệ trong đồ thị tri thức, sử dụng GNN và thuật toán HITS/PageRank.

* **Triển khai GNN**: Xây dựng mô hình **Graph Neural Network (GNN)** để học embedding cho các nút trong đồ thị (các đối tượng trong ảnh) và các mối quan hệ giữa chúng.
* **Tính toán độ tương đồng**: Sau khi học được các embedding cho các đối tượng, sử dụng **cosine similarity** để tính toán độ tương đồng giữa các đối tượng và ảnh, giúp tìm ra các ảnh có mối quan hệ gần gũi (ví dụ: "dog next to car").
* **Sử dụng HITS/PageRank**: Áp dụng các thuật toán **HITS** và **PageRank** để xác định các đối tượng quan trọng trong đồ thị, từ đó cải thiện độ chính xác khi tìm kiếm ảnh có mối quan hệ phức tạp.