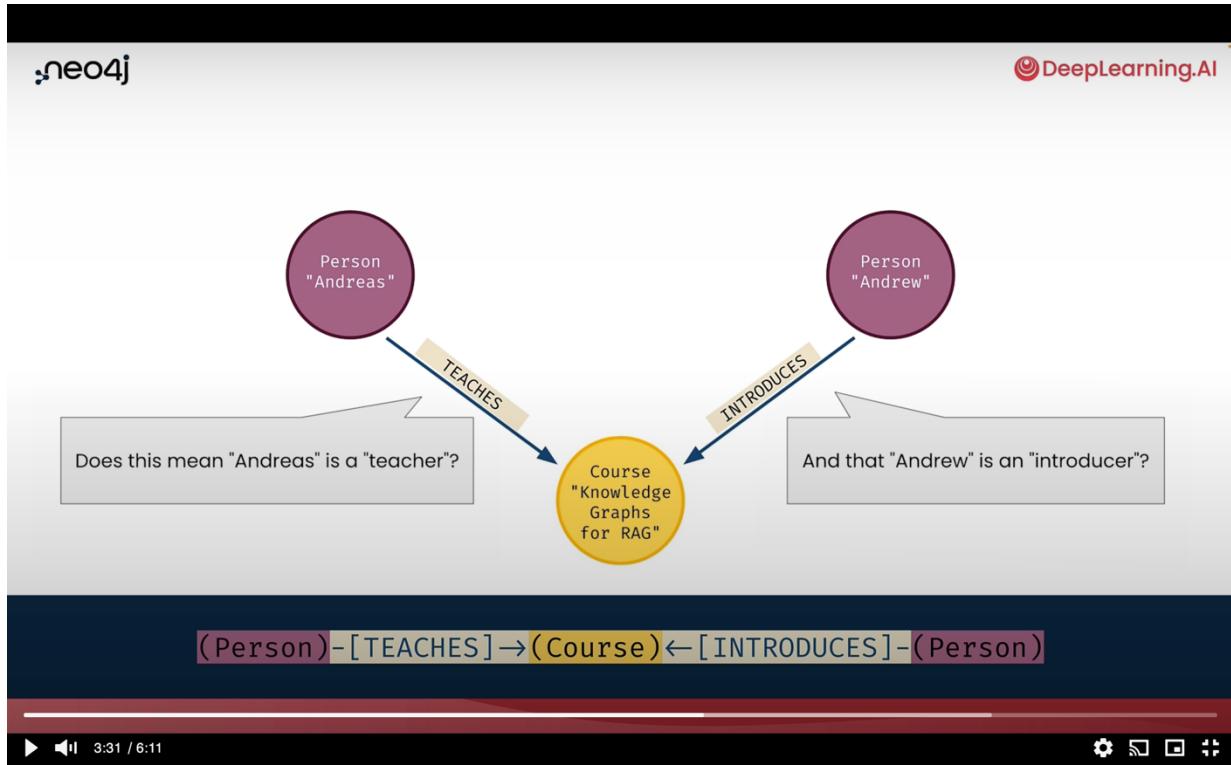


Course: <https://www.deeplearning.ai/short-courses/knowledge-graphs-rag/>

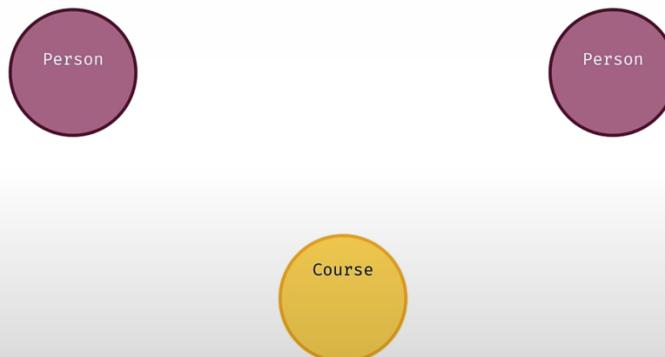
Certificate:

<https://learn.deeplearning.ai/accomplishments/c3ff9937-6c59-4dd4-9de4-95aad6d0300d?usp=sharing>

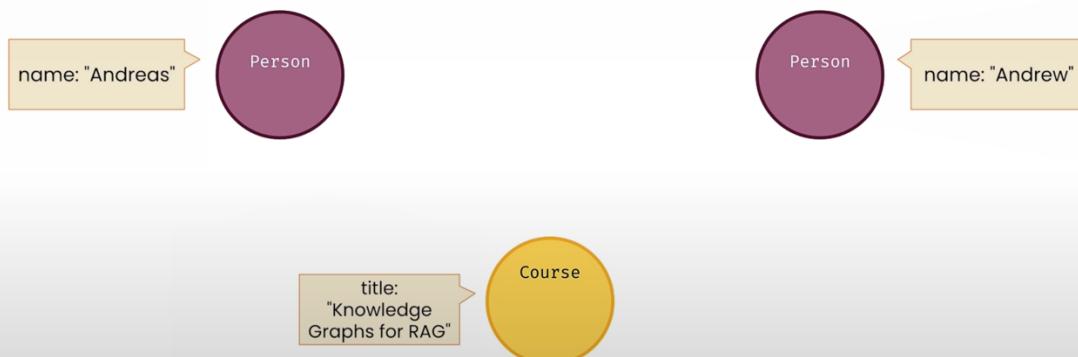
KG fundamentals:



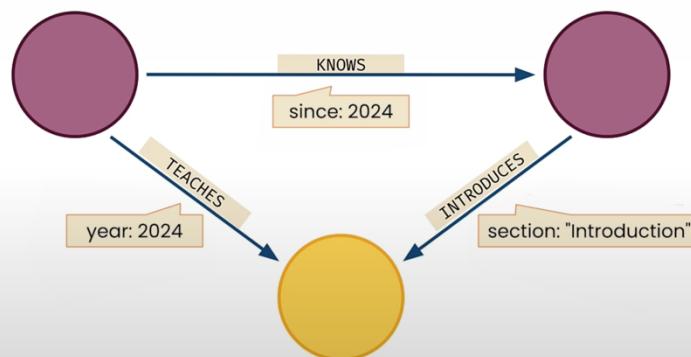
Formally, nodes have labels



Formally, nodes are data record  
with key/value properties



Relationships are also data records.  
They have a direction, a type, and properties



## What is a Knowledge Graph?

A knowledge graph is a database that stores information in **nodes** and **relationships**.

Both nodes and relationships can have **properties**.

Nodes can be given **labels** to group them together.

Relationships always have a **type** and a **direction**.

## Query KG:

Knowledge Graphs for RAG

- Introduction
- Knowledge Graph Fundamentals
- Querying Knowledge Graphs
- Preparing Text for RAG
- Constructing a Knowledge Graph from Text Documents
- Adding Relationships to the SEC Knowledge Graph
- Expanding the SEC Knowledge Graph
- Chatting with the Knowledge Graph
- Conclusion

Course Feedback

Community

In progress

12%

jupyter L2-query\_with\_cypher

```
from dotenv import load_dotenv
import os
from langchain_community.graphs import Neo4jGraph
# Warning control
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')
```

• Initialize a knowledge graph instance using LangChain's Neo4j integration

In [ ]:

```
kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD)
```

Movie Knowledge Graph - Person acted in movie

(Person)-[ACTED\_IN]→(Movie)

Next Lesson

jupyter L2-query\_with\_cypher

```
File Edit View Insert Cell Kernel Widgets Help
```

```
from dotenv import load_dotenv
import os
from langchain_community.graphs import Neo4jGraph
# Warning control
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')
```

• Initialize a knowledge graph instance using LangChain's Neo4j integration

In [ ]:

```
kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD)
```

Querying the movie knowledge graph

- Match all nodes in the graph

Node properties

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

```

Not Trusted | Python 3 (ipykernel) O
File Edit View Insert Cell Kernel Widgets Help
In [ ]: import os
from langchain_community.graphs import Neo4jGraph
# Warning control
import warnings
warnings.filterwarnings("ignore")
In [ ]: load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')

```

- Initialize a knowledge graph instance using LangChain's Neo4j integration

```
In [ ]: kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD)
```

**Querying the movie knowledge graph**

- Match all nodes in the graph

### All relationships between a Person and a Movie

```

graph LR
    Person((Person)) -- FOLLOWS --> Person
    Person -- ACTED_IN --> Movie((Movie))
    Person -- DIRECTED --> Movie
    Person -- WROTE --> Movie
    Person -- PRODUCED --> Movie
    Person -- REVIEWED --> Movie

```

[Next Lesson](#)

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

```

Not Trusted | Python 3 (ipykernel) O
File Edit View Insert Cell Kernel Widgets Help
In [ ]: cypher = """
MATCH (n)
RETURN count(n)
"""

```

```
In [ ]: result = kg.query(cypher)
result
```

```
In [ ]: cypher = """
MATCH (n)
RETURN count(n) AS numberOfRowsInSection
"""

```

```
In [ ]: result = kg.query(cypher)
result
```

```
In [ ]: print(f"There are {result[0]['numberOfNodes']} nodes in the graph")
```

- Match only the Movie nodes by specifying the node label

```

cypher = """
MATCH (n:Movie)
RETURN count(n)
"""

result = kg.query(cypher)
result

```

[Next Lesson](#)

Total 171 nodes in the graph.

Return is a dict, where key is what we ask in query.

We use cypher as query language for neo4j.

We can rename the key:

The screenshot shows a Jupyter notebook interface with the title "L2-query\_with\_cypher". The notebook contains the following code:

```

jupyter L2-query_with_cypher
File Edit View Insert Cell Kernel Widgets Help
Logout Python 3 (ipykernel) Not Trusted
Querying the movie knowledge graph
• Match all nodes in the graph
In [ ]: cypher = """
MATCH (n)
RETURN count(n)
"""
result = kg.query(cypher)
result
In [ ]: cypher = """
MATCH (n)
RETURN count(n) AS numberOfNodes
"""
result = kg.query(cypher)
result
In [ ]: print(f"There are {result[0]['numberOfNodes']} nodes in this graph.")

• Match only the Movie nodes by specifying the node label

```

The right panel shows the results of the first query:

```

cypher = """
MATCH (n)
RETURN count(n) AS numberOfNodes
"""

result = kg.query(cypher)
result
[{'numberOfNodes': 171}]

```

A red box highlights the result: `[{'numberOfNodes': 171}]`.

Now query based on nodes labels.

The screenshot shows a Jupyter notebook interface with the title "L2-query\_with\_cypher". The notebook contains the following code:

```

jupyter L2-query_with_cypher
File Edit View Insert Cell Kernel Widgets Help
Logout Python 3 (ipykernel) Not Trusted
Querying the movie knowledge graph
• Match only the Movie nodes by specifying the node label
In [ ]: cypher = """
MATCH (n:Movie)
RETURN count(n) AS numberOfMovies
"""
kg.query(cypher)

• Change the variable name in the node pattern match for improved readability
In [ ]: cypher = """
MATCH (m:Movie)
RETURN count(m) AS numberOfMovies
"""
kg.query(cypher)

• Match only the Person nodes

```

The right panel shows the results of the first query:

```

cypher = """
MATCH (n:Movie)
RETURN count(n) AS numberOfMovies
"""

result = kg.query(cypher)
result
[{'numberOfMovies': 38}]

```

A red box highlights the result: `[{'numberOfMovies': 38}]`.

Can change name of variable ,n to m

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

File Edit View Insert Cell Kernel Widgets Help Python 3 (ipykernel) Logout

```
In [ ]: result = kg.query(cypher)
result
```

```
In [ ]: print(f"There are {result[0]['numberOfNodes']} nodes in this graph.")
```

- Match only the Movie nodes by specifying the node label

```
In [ ]: cypher = """
MATCH (n:Movie)
RETURN count(n) AS numberOfMovies
"""
kg.query(cypher)
```

- Change the variable name in the node pattern match for improved readability

```
In [ ]: cypher = """
MATCH (:Movie)
RETURN count(m) AS numberOfMovies
"""
kg.query(cypher)
```

- Match only the Person nodes

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

File Edit View Insert Cell Kernel Widgets Help Python 3 (ipykernel) Logout

```
MATCH (:Movie)
RETURN count(m) AS numberOfMovies
"""
kg.query(cypher)
```

- Match only the Person nodes

```
In [ ]: cypher = """
MATCH (people:Person)
RETURN count(people) AS numberOfPeople
"""
kg.query(cypher)
```

- Match a single person by specifying the value of the `name` property on the Person node

```
In [ ]: cypher = """
MATCH (tom:Person {name:"Tom Hanks"})
RETURN tom
"""
kg.query(cypher)
```

- Match a single Movie by specifying the value of the `title` property

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

jupyter L2-query\_with\_cypher

File Edit View Insert Cell Kernel Widgets Help

In [ ]: cypher = """  
MATCH (m:Movie)  
RETURN count(m) AS numberOfRowsMovies  
"""  
kg.query(cypher)

• Match only the Person nodes

In [ ]: cypher = """  
MATCH (people:Person)  
RETURN count(people) AS numberOfRowsPeople  
"""  
kg.query(cypher)

• Match a single person by specifying the value of the name property on the Person node

In [ ]: cypher = """  
MATCH (tom:Person {name:"Tom Hanks"})  
RETURN tom  
"""  
kg.query(cypher)

• Match a single Movie by specifying the value of the title property

In [ ]: cypher = """  
MATCH (m:Movie {title:"Cloud Atlas"})  
RETURN m  
"""  
kg.query(cypher)

cypher = """  
MATCH (n)  
RETURN count(n) AS numberOfRowsNodes  
"""  
  
result = kg.query(cypher)  
result  
  
[{'numberOfNodes': 171}]  
  
print(f"There are {result[0]['numberOfNodes']} nodes in this graph.")  
There are 171 nodes in this graph.  
  
cypher = """  
MATCH (people:Person)  
RETURN count(people) AS numberOfRowsPeople  
"""  
  
kg.query(cypher)  
  
[{'numberOfPeople': 133}]  
  
cypher = """  
MATCH (tom:Person {name:"Tom Hanks"})  
RETURN tom  
"""  
  
kg.query(cypher)  
  
[{'tom': {'born': 1956, 'name': 'Tom Hanks'}}]

8:57 / 19:02

My Learnings

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

jupyter L2-query\_with\_cypher

File Edit View Insert Cell Kernel Widgets Help

In [ ]: cypher = """  
MATCH (m:Movie)  
RETURN count(m) AS numberOfRowsMovies  
"""  
kg.query(cypher)

• Match only the Person nodes

In [ ]: cypher = """  
MATCH (people:Person)  
RETURN count(people) AS numberOfRowsPeople  
"""  
kg.query(cypher)

• Match a single person by specifying the value of the name property on the Person node

In [ ]: cypher = """  
MATCH (tom:Person {name:"Tom Hanks"})  
RETURN tom  
"""  
kg.query(cypher)

• Match a single Movie by specifying the value of the title property

In [ ]: cypher = """  
MATCH (m:Movie {title:"Cloud Atlas"})  
RETURN m  
"""  
kg.query(cypher)

cypher = """  
MATCH (tom:Person {name:"Tom Hanks"})  
RETURN tom  
"""  
  
kg.query(cypher)  
  
[{'tom': {'born': 1956, 'name': 'Tom Hanks'}}]  
  
cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas  
"""  
  
kg.query(cypher)  
  
[{'cloudAtlas': {'tagline': 'Everything is connected',  
'title': 'Cloud Atlas',  
'released': 2012}}]

9:35 / 19:02

My Learnings

Next Lesson

The screenshot shows a Jupyter Notebook interface on a web browser. The title bar says "jupyter L2-query\_with\_cypher". The left pane contains code cells and text sections. The right pane shows the results of the queries.

**Code Cells (Left):**

- In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released  
"""  
kg.query(cypher)
- In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released, cloudAtlas.tagline  
"""  
kg.query(cypher)
- Cypher patterns with conditional matching**
- In [ ]: cypher = """  
MATCH (nineties:Movie)  
WHERE nineties.released >= 1990  
AND nineties.released < 2000  
RETURN nineties.title  
"""

**Results (Right):**

```
kg.query(cypher)
[{"cloudAtlas": {"tagline": "Everything is connected",
  "title": "Cloud Atlas",
  "released": 2012}}]

cypher = """
MATCH (cloudAtlas:Movie {title:'Cloud Atlas'})
RETURN cloudAtlas.released
"""

kg.query(cypher)
[{"cloudAtlas.released": 2012}]

kg.query(cypher)
[{"cloudAtlas": {"tagline": "Everything is connected"}]
```

Bottom right: Next Lesson

The screenshot shows a Jupyter Notebook interface on a web browser. The title bar says "jupyter L2-query\_with\_cypher". The left pane contains code cells and text sections. The right pane shows the results of the queries.

**Code Cells (Left):**

- In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released  
"""  
kg.query(cypher)
- In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released, cloudAtlas.tagline  
"""  
kg.query(cypher)
- Cypher patterns with conditional matching**
- In [ ]: cypher = """  
MATCH (nineties:Movie)  
WHERE nineties.released >= 1990  
AND nineties.released < 2000  
RETURN nineties.title  
"""

**Results (Right):**

```
kg.query(cypher)
[{"cloudAtlas": {"tagline": "Everything is connected",
  "title": "Cloud Atlas",
  "released": 2012}}]

cypher = """
MATCH (cloudAtlas:Movie {title:'Cloud Atlas'})
RETURN cloudAtlas.released
"""

kg.query(cypher)
[{"cloudAtlas.released": 2012}]

kg.query(cypher)
[{"cloudAtlas": {"tagline": "Everything is connected"}]
```

Bottom right: Next Lesson

Query range:

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

Logout

File Edit View Insert Cell Kernel Widgets Help

In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released  
"""  
kg.query(cypher)

• Return only the released property of the matched Movie node

In [ ]: cypher = """  
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})  
RETURN cloudAtlas.released, cloudAtlas.tagline  
"""  
kg.query(cypher)

• Return two properties

**Cypher patterns with conditional matching**

In [ ]: cypher = """  
MATCH (nineties:Movie)  
WHERE nineties.released >= 1990  
AND nineties.released < 2000  
RETURN nineties.title  
"""

cypher = """  
MATCH (nineties:Movie)  
WHERE nineties.released >= 1990  
AND nineties.released < 2000  
RETURN nineties.title  
"""

kg.query(cypher)|

```
[{'nineties.title': 'The Matrix'},  
{'nineties.title': "The Devil's Advocate"},  
{'nineties.title': 'A Few Good Men'},  
{'nineties.title': 'As Good as It Gets'},  
{'nineties.title': 'What Dreams May Come'},  
{'nineties.title': 'Snow Falling on Cedars'},  
{'nineties.title': "You've Got Mail"},  
{'nineties.title': 'Sleepless in Seattle'},  
{'nineties.title': 'Joe Versus the Volcano'},  
{'nineties.title': 'When Harry Met Sally'},  
{'nineties.title': 'That Thing You Do'},  
{'nineties.title': 'The Birdcage'},  
{'nineties.title': 'Unforgiven'},  
{'nineties.title': 'Johnny Mnemonic'},  
{'nineties.title': 'The Green Mile'},  
{'nineties.title': 'Hoffa'},  
{'nineties.title': 'Apollo 13'},  
{'nineties.title': 'Twister'},  
{'nineties.title': 'Bicentennial Man'},  
{'nineties.title': 'A League of Their Own'}]
```

11:48 / 19:02

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

Logout

File Edit View Insert Cell Kernel Widgets Help

In [ ]: cypher = """  
MATCH (actor:Person)-[:ACTED\_IN]->(movie:Movie)  
RETURN actor.name, movie.title LIMIT 10  
"""  
kg.query(cypher)

In [ ]: cypher = """  
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED\_IN]->(tom)  
RETURN tom.name, tomMovies.title  
"""  
kg.query(cypher)

In [ ]: cypher = """  
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED\_IN]->(m)->  
RETURN coActors.name, m.title  
"""  
kg.query(cypher)

**Pattern matching with multiple nodes**

Pattern matching with multiple nodes

cypher = """  
MATCH (actor:Person)-[:ACTED\_IN]->(movie:Movie)  
RETURN actor.name, movie.title LIMIT 10  
"""  
kg.query(cypher)|

```
[{'actor.name': 'Emil Eifrem', 'movie.title': 'The Matrix'},  
{'actor.name': 'Hugo Weaving', 'movie.title': 'The Matrix'},  
{'actor.name': 'Laurence Fishburne', 'movie.title': 'The Matrix'},  
{'actor.name': 'Carrie-Anne Moss', 'movie.title': 'The Matrix'},  
{'actor.name': 'Keanu Reeves', 'movie.title': 'The Matrix'},  
{'actor.name': 'Hugo Weaving', 'movie.title': 'The Matrix Reloaded'},  
{'actor.name': 'Laurence Fishburne', 'movie.title': 'The Matrix Reloaded'},  
{'actor.name': 'Carrie-Anne Moss', 'movie.title': 'The Matrix Reloaded'},  
{'actor.name': 'Keanu Reeves', 'movie.title': 'The Matrix Reloaded'},  
{'actor.name': 'Hugo Weaving', 'movie.title': 'The Matrix Revolutions'}
```

12:55 / 19:02

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

Pattern matching with multiple nodes

```
In [ ]: cypher = """  
    MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)  
    RETURN actor.name, movie.title LIMIT 10  
    """  
kg.query(cypher)
```

```
In [ ]: {name: "Tom Hanks"}-[:ACTED_IN]->(tomHanksMovies:Movie)  
tomHanksMovies.title
```

```
In [ ]: {name:"Tom Hanks"})-[:ACTED_IN]->(m)-<[:ACTED_IN]-(coActors)  
m.title
```

Delete data from the graph

```
In [ ]: cypher = """  
    MATCH (emil:Person {name:"Emil Eifrem"})-[actedIn:ACTED_IN]->  
    (m)  
    DELETE m  
    """  
kg.query(cypher)
```

```
In [ ]: cypher = """  
    MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies:Movie)  
    RETURN tom.name,tomHanksMovies.title  
    """  
kg.query(cypher)
```

```
In [ ]: cypher = """  
    MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)  
    RETURN actor.name, movie.title LIMIT 10  
    """  
kg.query(cypher)
```

```
In [ ]: {name: "Tom Hanks"}-[:ACTED_IN]->(tomHanksMovies:Movie)  
tomHanksMovies.title
```

```
In [ ]: {name:"Tom Hanks"})-[:ACTED_IN]->(m)-<[:ACTED_IN]-(coActors)  
m.title
```

Next Lesson

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs

### jupyter L2-query\_with\_cypher

Pattern matching with multiple nodes

```
In [ ]: cypher = """  
    MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)  
    RETURN actor.name, movie.title LIMIT 10  
    """  
kg.query(cypher)
```

```
In [ ]: {name: "Tom Hanks"}-[:ACTED_IN]->(tomHanksMovies:Movie)  
tomHanksMovies.title
```

```
In [ ]: {name:"Tom Hanks"})-[:ACTED_IN]->(m)-<[:ACTED_IN]-(coActors)  
m.title
```

Delete data from the graph

```
In [ ]: cypher = """  
    MATCH (emil:Person {name:"Emil Eifrem"})-[actedIn:ACTED_IN]->  
    (m)  
    DELETE m  
    """  
kg.query(cypher)
```

```
In [ ]: cypher = """  
    MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies:Movie)  
    RETURN tom.name,tomHanksMovies.title  
    """  
kg.query(cypher)
```

```
In [ ]: cypher = """  
    MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)  
    RETURN actor.name, movie.title LIMIT 10  
    """  
kg.query(cypher)
```

```
In [ ]: {name: "Tom Hanks"}-[:ACTED_IN]->(tomHanksMovies:Movie)  
tomHanksMovies.title
```

```
In [ ]: {name:"Tom Hanks"})-[:ACTED_IN]->(m)-<[:ACTED_IN]-(coActors)  
m.title
```

Next Lesson

**Delete data from the graph**

```
In [ ]:
cypher = """
MATCH (emil:Person {name:"Emil Eifrem"})-[actedIn:ACTED_IN]->(movie:Movie)
RETURN emil.name, movie.title
"""

kg.query(cypher)

[{"emil.name": "Emil Eifrem", "movie.title": "The Matrix"}]
```

**Adding data to the graph**

```
In [ ]:
cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)

In [ ]:
cypher = """
MATCH (andreas:Person {name:"Andreas"})-[:ACTED_IN]->(emil:Person {name:"Emil Eifrem"})
DELETE actedin
"""

kg.query(cypher)
```

[Next Lesson](#)

## Delete:

**Delete data from the graph**

```
In [ ]:
cypher = """
MATCH (emil:Person {name:"Emil Eifrem"})-[actedIn:ACTED_IN]->(movie:Movie)
RETURN emil.name, movie.title
"""

kg.query(cypher)

[{"emil.name": "Emil Eifrem", "movie.title": "The Matrix"}]
```

**Adding data to the graph**

```
In [ ]:
cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)

In [ ]:
cypher = """
MATCH (andreas:Person {name:"Andreas"})-[:ACTED_IN]->(emil:Person {name:"Emil Eifrem"})
DELETE actedin
"""

kg.query(cypher)
```

[Next Lesson](#)

## Adding:

```

jupyter L2-query_with_cypher
File Edit View Insert Cell Kernel Widgets Help
Logout Python 3 (ipykernel) Not Trusted

In [ ]:
on {name:"Andreas"}-[:ACTED_IN]->(movie:Movie)

Adding data to the graph
In [ ]:
cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)

In [ ]:
cypher = """
MATCH (andreas:Person {name:"Andreas"}), (emil:Person {name:"Emil Eifrem"})
MERGE (andreas)-[hasRelationship:WORKS_WITH]->(emil)
RETURN andreas, hasRelationship, emil
"""
kg.query(cypher)

```

Querying data to the graph:

```

cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)
[{'andreas': {'name': 'Andreas'}}]


```

Next Lesson

```

jupyter L2-query_with_cypher
File Edit View Insert Cell Kernel Widgets Help
Logout Python 3 (ipykernel) Not Trusted

In [ ]:
on {name:"Andreas"}-[:ACTED_IN]->(movie:Movie)

Adding data to the graph
In [ ]:
cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)

In [ ]:
cypher = """
MATCH (andreas:Person {name:"Andreas"}), (emil:Person {name:"Emil Eifrem"})
MERGE (andreas)-[hasRelationship:WORKS_WITH]->(emil)
RETURN andreas, hasRelationship, emil
"""
kg.query(cypher)

```

Querying data to the graph:

```

cypher = """
CREATE (andreas:Person {name:"Andreas"})
RETURN andreas
"""

kg.query(cypher)
[{'andreas': {'name': 'Andreas'}}]

cypher = """
MATCH (andreas:Person {name:"Andreas"}), (emil:Person {name:"Emil Eifrem"})
MERGE (andreas)-[hasRelationship:WORKS_WITH]->(emil)
RETURN andreas, hasRelationship, emil
"""

kg.query(cypher)
[{'andreas': {'name': 'Andreas'}, 'hasRelationship': {'name': 'Emil Eifrem'}, 'WORKS_WITH': {'born': 1978, 'name': 'Emil Eifrem'}, 'emil': {'born': 1978, 'name': 'Emil Eifrem'}}


```

Next Lesson

Here, we first find individual persons, without any relationship, hence comma in match. Then we use MERGE, merge is similar to create but if the relationship already exists, it won't create it again. Hence merge.

NEXT CHAPTER:

## Lesson 3: Preparing Text Data for RAG

The screenshot shows a Jupyter Notebook interface within a web browser window. The title bar reads "learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/4/preparing-text-for-rag". The notebook has two cells visible:

```
import os
from langchain_community.graphs import Neo4jGraph

# Load from environment
load_dotenv('../.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD, database=NEO4J_DATABASE
)

kg.query("""
    CREATE VECTOR INDEX movie_tagline_embeddings IF NOT EXISTS
    FOR (:Movie) ON (.taglineEmbedding)
    OPTIONS { indexConfig: {
        'vector.dimensions': 1536,
        'vector.similarity_function': 'cosine'
    } }""")
```

The right side of the interface shows a file tree labeled "Project" and a "Pull Requests" section.

Creating a vector index if not already exist.

Index on tagline

Adding some configs.

The screenshot shows a Jupyter Notebook interface within a browser window. The title bar says "DeepLearning.AI Beta". The notebook has a single cell containing Python code. The code imports dotenv and langchain\_community graphs, sets up Neo4j with environment variables, and connects to a Neo4j graph instance. It then runs a Cypher query to show vector indexes and a JSON response defining a vector index for movie taglines.

```

import packages and set up Neo4j
In [ ]: from dotenv import load_dotenv
        import os
        from langchain_community.graphs import Neo4j
        # Warning control
        import warnings
        warnings.filterwarnings("ignore")
In [ ]: # Load from environment
        load_dotenv(override=True)
        NEO4J_URI = os.getenv('NEO4J_URI')
        NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
        NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
        NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')
        OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
        OPENAI_ENDPOINT = os.getenv('OPENAI_BASE_URL')
In [ ]: # Connect to the knowledge graph instance
        kg = Neo4jGraph(
            url=NEO4J_URI, username=NEO4J_USERNAME,
    
```

```

        `vector.dimensions': 1536,
         'vector.similarity_function': 'cosine'
     }"""
)
[]

kg.query("""
    SHOW VECTOR INDEXES
"""
)

[{"id": 3,
"name": "movie_tagline_embeddings",
"state": "ONLINE",
"populationPercent": 100.0,
"type": "VECTOR",
"entityType": "NODE",
"labelsOrTypes": ["Movie"],
"properties": ["taglineEmbedding"],
"indexProvider": "vector-1.0",
"owningConstraint": None,
"lastRead": None,
"readCount": 0}]

```

Running the openai encoder to get embeddings and store them as a vector index.  
In the cypher query, we can pass a value as params.

We directly run the code to generate the encodings within the query, to do it for all taglines for all movies.

The screenshot shows a Jupyter Notebook cell with the title "Populate the vector index". The code uses the `kg.query` method to run a Cypher query. The query matches movies where tagline is not null, encodes the tagline using an OpenAI API key, and creates a vector property for each movie. It also includes a separate query to return the tagline and its embedding for the first movie.

```

Populate the vector index
kg.query("""
    MATCH (movie:Movie) WHERE movie.tagline IS NOT NULL
    WITH movie, genai.vector.encode(
        movie.tagline,
        "OpenAI",
        {token: $openAiApiKey}
    ) AS vector
    CALL db.create.setNodeVectorProperty(movie, "taglineEmbedding", vector)
    """,
    params={"openAiApiKey":OPENAI_API_KEY}
)
[]

result = kg.query("""
    MATCH (m:Movie)
    WHERE m.tagline IS NOT NULL
    RETURN m.tagline, m.taglineEmbedding
    LIMIT 1
"""
)

```

Looking at the stored embeddings:

```
CALL db.create.setNodeVectorProperty(movie, "taglineEmbedding", vector)
      """,
      params={"openAiApiKey":OPENAI_API_KEY} )
```

```
[]
result = kg.query("""
    MATCH (m:Movie)
    WHERE m.tagline IS NOT NULL
    RETURN m.tagline, m.taglineEmbedding
    LIMIT 1
    """
)
```

```
result[0]['m.tagline']
'Welcome to the Real World'
```

```
result[0]['m.taglineEmbedding'][::10]
```

```
[0.017327824607491493,
-0.005428072530776262,
-0.001854208530858159,
-0.025637304410338402,
-0.014374218881130219,
0.016540195792913437,
-0.01693401113152504,
0.0003212047158740461,
-0.025099091231822968,
-0.029614828526973724]
```

```
len(result[0]['m.taglineEmbedding'])
```

```
1536
```

Query:

```
question = "What movies are about love?"
```

```
kg.query("""
    WITH genai.vector.encode(
        $question,
        "OpenAI",
        {token: $openAiApiKey}) AS question_embedding
    CALL db.index.vector.queryNodes(
        'movie_tagline_embeddings',
        $top_k,
        question_embedding
    ) YIELD node AS movie, score
    RETURN movie.title, movie.tagline, score
    """
    ,params={"openAiApiKey":OPENAI_API_KEY,
        "question": question,
        "top_k": 5
    })
```

```
[{'movie.title': 'Joe Versus the Volcano',
'movie.tagline': 'A story of love, lava and burning desire.',
'score': 0.906736433506012},
{'movie.title': 'As Good as It Gets',
'movie.tagline': 'A comedy from the heart that goes for the throat.',
'score': 0.9027702212333679},
{'movie.title': 'Snow Falling on Cedars',
'movie.tagline': 'First loves last. Forever.',
'score': 0.9016978740692139},
{'movie.title': 'When Harry Met Sally',
'movie.tagline': 'Can two friends sleep together and still love each other in the morning?',
'score': 0.8948053121566772},
{'movie.title': 'Sleepless in Seattle',
'movie.tagline': 'What if someone you never met... someone you never saw someone you never knew was the only someone for you?',
'score': 0.8947287797927856}]
```

# Lesson 4: Constructing a Knowledge Graph from Text Documents

## Cleaning:

Converted to json at the end.

The screenshot shows a Jupyter Notebook slide with the following content:

### Data cleaning

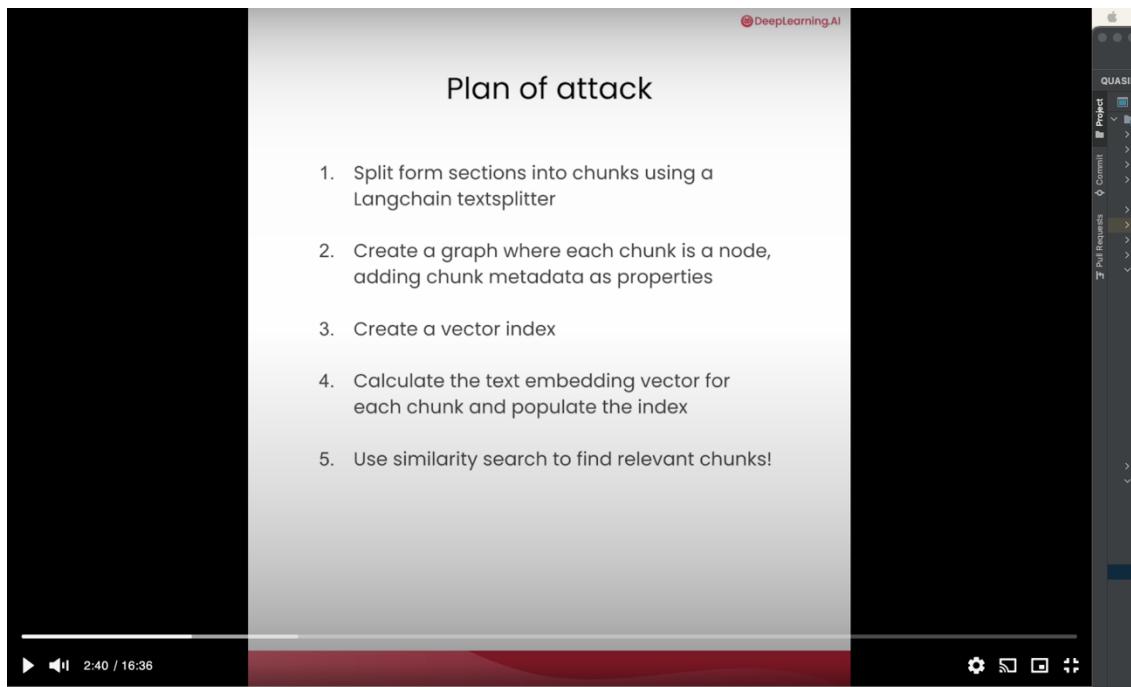
Completed 10-K forms are available to download as text files that contain XML components

In order to work with them, the following cleaning steps were applied

- Cleaned up the files using regex
- Parsed XML into python data structures using BeautifulSoup
- Extracted CIK (Central Index Key) ID which is a company identifier used by the SEC
- Extracted specific sections of the form (Items 1, 1a, 7, and 7a)

You can look in the data directory in the notebook if you'd like to examine the cleaned data for yourself.

At the bottom of the slide, there is a red navigation bar with icons for back, forward, and search, along with a progress bar showing 2:08 / 16:36. On the right side of the slide, there is a sidebar with project and commit history.



```
from dotenv import load_dotenv
import os

# Common data processing
import json
import textwrap

# Langchain
from langchain_community.graphs import Neo4jGraph
from langchain_community.vectorstores import Neo4jVector
from langchain_openai import OpenAIEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains import RetrievalQAWithSourcesChain
from langchain_openai import ChatOpenAI

# Load from environment
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE') or 'neo4j'
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

# Global constants
VECTOR_INDEX_NAME = 'form_10k_chunks'
VECTOR_NODE_LABEL = 'Chunk'
VECTOR_SOURCE_PROPERTY = 'text'
VECTOR_EMBEDDING_PROPERTY = 'textEmbedding'
```

The code editor shows a Python script with imports for dotenv, os, json, textwrap, Neo4jGraph, Neo4jVector, OpenAIEmbeddings, RecursiveCharacterTextSplitter, RetrievalQAWithSourcesChain, and ChatOpenAI. It loads environment variables from .env and defines global constants for a vector index named 'form\_10k\_chunks'.

The bottom left corner shows a progress bar at 3:18 / 16:36. The bottom right corner shows standard video control icons.

```
first_file_name = "./data/form10k/0000950170-23-027948.json"

first_file_as_object = json.load(open(first_file_name))

type(first_file_as_object)

dict

for k,v in first_file_as_object.items():
    print(k, type(v))

item1 <class 'str'>
item1a <class 'str'>
item7 <class 'str'>
item7a <class 'str'>
cik <class 'str'>
cusip6 <class 'str'>
cusip <class 'list'>
names <class 'list'>
source <class 'str'>

item1_text = first_file_as_object['item1']

item1_text[0:1500]

'>Item 1. \nBusiness\n\n\nOverview\n\n\nNetApp, Inc. (NetApp, we, us or the Company) is a global cloud-led, data-centric software company. We were incorporated in 1992 and are headquartered in San Jose, California. Building on more than three decades of innovation, we give customers the freedom to manage applications and data across hybrid multicloud environments. Our portfolio of cloud services, and storage infrastructure, powered by intelligent data management software, enables applications to run faster, more reliably, and more securely, all at a lower cost.\n\nOur opportunity is defined by the durable megatrends of data-driven digital and cloud transformations. NetApp helps organizations meet the complexities created by rapid data and cloud growth, multi-cloud management, and the adoption of next-generation technologies, such as AI, Kubernetes, and modern databases. Our modern approach to hybrid, multicloud infrastructure and data management, which we term 'evolved cloud', provides customers the ability to leverage data across their entire estate with simplicity, security, and sustainability which increases our relevance and value to our customers.\n\n\nan evolv
```

## Creating chunks, using langchain text splitter.

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 2000,
    chunk_overlap = 200,
    length_function = len,
    is_separator_regex = False,
)

item1_text_chunks = text_splitter.split_text(item1_text)

type(item1_text_chunks)

list

len(item1_text_chunks)

254

item1_text_chunks[0]

'>Item 1. \nBusiness\n\n\nOverview\n\n\nNetApp, Inc. (NetApp, we, us or the Company) is a global cloud-led, data-centric software company. We were incorporated in 1992 and are headquartered in San Jose, California. Building on more than three decades of innovation, we give customers the freedom to manage applications and data across hybrid multicloud environments. Our portfolio of cloud services, and storage infrastructure, powered by intelligent data management software, enables applications to run faster, more reliably, and more securely, all at a lower cost.\n\nOur opportunity is defined by the durable megatrends of data-driven digital and cloud transformations. NetApp helps organizations meet the complexities created by rapid data and cloud growth, multi-cloud management, and the adoption of next-generation technologies, such as AI, Kubernetes, and modern databases. Our modern approach to hybrid, multicloud infrastructure and data management, which we term 'evolved cloud', provides customers the ability to leverage data across their entire estate with simplicity, security, and sustainability which increases our relevance and value to our customers.\n\n\nan evolv
```

Helper function to parse the data, create chunks and return chunks + metadata.

```
def split_form10k_data_from_file(file):
    chunks_with_metadata = [] # use this to accumulate chunk records
    file_as_object = json.load(open(file)) # open the json file
    for item in ['item1', 'item1a', 'item7', 'item7a']: # pull these keys
        print(f'Processing {item} from {file}')
        item_text = file_as_object[item] # grab the text of the item
        item_text_chunks = text_splitter.split_text(item_text) # split
        chunk_seq_id = 0
        for chunk in item_text_chunks[:20]: # only take the first 20
            form_id = file[file.rindex('/') + 1:file.rindex('.')] # ex
            # finally, construct a record with metadata and the chunk
            chunks_with_metadata.append({
                'text': chunk,
                # metadata from looping...
                'f10kItem': item,
                'chunkSeqId': chunk_seq_id,
                # constructed metadata...
                'formId': f'{form_id}', # pulled from the filename
                'chunkId': f'{form_id}-{item}-{chunk(chunk_seq_id:04d)}',
                # metadata from file...
                'names': file_as_object['names'],
                'cik': file_as_object['cik'],
                'cusip6': file_as_object['cusip6'],
                'source': file_as_object['source'],
            })
            chunk_seq_id += 1
        print(f'\tSplit into {chunk_seq_id} chunks')
    return chunks_with_metadata
```

▶️ 6:57 / 16:36

```
first_file_chunks = split_form10k_data_from_file(first_file_name)
Processing item1 from ./data/form10k/0000950170-23-027948.json
    Split into 20 chunks
Processing item1a from ./data/form10k/0000950170-23-027948.json
    Split into 1 chunks
Processing item7 from ./data/form10k/0000950170-23-027948.json
    Split into 1 chunks
Processing item7a from ./data/form10k/0000950170-23-027948.json
    Split into 1 chunks
first_file_chunks[0]
{
    "text": ">Item 1. \nBusiness\n\nOverview\n\n\nNetApp, Inc. (NetApp, we, us or the Company) is a global cloud-led, data-centric software company. We were incorporated in 1992 and are headquartered in San Jose, California. Building on more than three decades of innovation, we give customers the freedom to manage applications and data across hybrid multicloud environments. Our portfolio of cloud services, and storage infrastructure, powered by intelligent data management software, enables applications to run faster, more reliably, and more securely, all at a lower cost.\n\nOur opportunity is defined by the durable megatrends of data-driven digital and cloud transformations. NetApp helps organizations meet the complexities created by rapid data and cloud growth, multi-cloud management, and the adoption of next-generation technologies, such as AI, Kubernetes, and modern databases. Our modern approach to hybrid, multicloud infrastructure and data management, which we term 'evolved cloud', provides customers the ability to leverage data across their entire estate with simplicity, security, and sustainability which increases our relevance and value to our customers.\n\nIn an evolved cloud state, the cloud is fully integrated into an organization's architecture and operations. Data centers and clouds are seamlessly united and hybrid multicloud operations are simplified, with consistency and observability across environments. The key benefits NetApp brings to an organization's hybrid multicloud environment are:\n\n\nOperational simplicity: NetApp's use of open source, open architectures and APIs, microservices, and common capabilities and data services facilitate the creation of applications that can run anywhere.\n\n"
}
```

▶️ 7:26 / 16:36

## Adding chunk to KG as a node.

**Creating another index: which makes a unique constraint on chunk id so that it doesn't repeat.**

```

merge_chunk_node_query = """
MERGE(mergedChunk:Chunk {chunkId: $chunkParam.chunkId})
ON CREATE SET
    mergedChunk.names = $chunkParam.names,
    mergedChunk.formId = $chunkParam.formId,
    mergedChunk.cik = $chunkParam.cik,
    mergedChunk.cusip6 = $chunkParam.cusip6,
    mergedChunk.source = $chunkParam.source,
    mergedChunk.f10kItem = $chunkParam.f10kItem,
    mergedChunk.chunkSeqId = $chunkParam.chunkSeqId,
    mergedChunk.text = $chunkParam.text
RETURN mergedChunk
"""

kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD, d
)

kg.query(merge_chunk_node_query,
         params={'chunkParam':first_file_chunks[0]})
```

[{'mergedChunk': {'formId': '0000950170-23-027948',
 'f10kItem': 'item1',
 'names': ['Netapp Inc', 'NETAPP INC'],
 'cik': '1002047',
 'cusip6': '64110D',
 'source': 'https://www.sec.gov/Archives/edgar/data/1002047/000095017023027948/0000950170-23-027948-index.htm',
 'text': '>Item 1. \nBusiness\n\nOverview\n\nNetApp, Inc. (NetApp, we, us or the Company) is a global cloud-led, data-centric software company. We were incorporated in 1992 and are headquartered in San Jose, California. Building on more than three decades of innovation, we give customers the freedom to manage applications and data across hybrid multicloud environments. Our portfolio of cloud services, and storage infrastructure, powered by intelligent data management software, enables applications to run faster, more reliably, and more securely, al'}

▶ ⏸ 8:59 / 16:36

## Adding all chunks as node.

```

node_count = 0
for chunk in first_file_chunks:
    print("Creating ':Chunk' node for chunk ID {chunk['chunkId']}")"
    kg.query(merge_chunk_node_query,
             params={
                 'chunkParam': chunk
             })
    node_count += 1
print(f"Created {node_count} nodes")
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
00
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
01
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
02
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
03
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
04
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
05
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
06
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
07
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
08
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
09
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
10
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
11
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
12
Creating ':Chunk' node for chunk ID 0000950170-23-027948-item1-chunk00
```

▶ ⏸ 10:14 / 16:36

**Creating vector index to store embeddings of text in these chunks:**

```
kg.query("""  
CREATE VECTOR INDEX `form_10k_chunks` IF NOT EXISTS  
FOR (c:Chunk) ON (c.textEmbedding)  
OPTIONS { indexConfig: {  
    'vector.dimensions': 1536,  
    'vector.similarity_function': 'cosine'  
}}  
""")  
[]  
  
kg.query("SHOW INDEXES")  
[{"id": 0,  
 "name": "form_10k_chunks",  
 "state": "ONLINE",  
 "populationPercent": 100.0,  
 "type": "VECTOR",  
 "entityType": "NODE",  
 "labelsOrTypes": ["Chunk"],  
 "properties": ["textEmbedding"],  
 "indexProvider": "vector-1.0",  
 "owningConstraint": None,  
 "lastRead": None,  
 "readCount": 0},  
 {"id": 1,  
 "name": "index_343aff4e",  
 "state": "ONLINE",  
 "populationPercent": 100.0,  
 "type": "LOOKUP",  
 "entityType": "NODE",  
 "labelsOrTypes": None,  
 "properties": None,  
 "indexProvider": "token-lookup-1.0",  
 "owningConstraint": None,  
 "lastRead": neo4j.time.DateTime(2024, 3, 12, 15, 21, 55, 173000000, 0)}]
```

▶ 🔍 11:07 / 16:36

## creating embeddings and storing in nodes:

```
kg.query("""  
MATCH (chunk:Chunk) WHERE chunk.textEmbedding IS NULL  
WITH chunk, genai.vector.encode(chunk.text, "OpenAI", {token: $openAiApiKey}) AS vector  
CALL db.create.setNodeVectorProperty(chunk, "textEmbedding", vector)  
""",  
params={"openAiApiKey":OPENAI_API_KEY} )|
```

▶ 🔍 11:28 / 16:36

```
MATCH (chunk:Chunk) WHERE chunk.textEmbedding IS NULL
WITH chunk, genai.vector.encode(chunk.text, "OpenAI", {token: $openAiApiKey})
CALL db.create.setNodeVectorProperty(chunk, "textEmbedding", vector)
    params={"openAiApiKey":OPENAI_API_KEY} )
[]

kg.refresh_schema()
print(kg.schema)

Node properties are the following:
Chunk {textEmbedding: LIST, f10kItem: STRING, chunkSeqId: INTEGER, text: STRING, cik: STRING, cusip6: STRING, chunkId: STRING, names: LIST, formId: STRING, source: STRING}
Relationship properties are the following:

The relationships are the following:
```

## Question embedding and searching:

```
def neo4j_vector_search(question):
    """Search for similar nodes using the Neo4j vector index"""
    vector_search_query = """
        WITH genai.vector.encode($question, "OpenAI", {token: $openAiApiKey})
        CALL db.index.vector.queryNodes($index_name, $top_k, question_embedding)
        RETURN score, node.text AS text
    """
    similar = kg.query(vector_search_query,
                        params={
                            'question': question,
                            'openAiApiKey':OPENAI_API_KEY,
                            'index_name':VECTOR_INDEX_NAME,
                            'top_k': 10
                        })
    return similar

search_results = neo4j_vector_search(
    'In a single sentence, tell me about Netapp.'
)

search_results[0]
```

```
{'score': 0.9356340169906616,
'text': '>Item 1. \nBusiness\n\nOverview\n\nNetApp, Inc. (NetApp, we, us or the Company) is a global cloud-led, data-centric software company. We were incorporated in 1992 and are headquartered in San Jose, California. Building on more than three decades of innovation, we give customers the freedom to manage applications and data across hybrid multicloud environments. Our portfolio of cloud services, and storage infrastructure, powered by intelligent data management software, enables applications to run faster, more reliably, and more securely, all at a lower cost.\n\nOur opportunity is defined by the durable megatrends of data-driven digital and cloud transformations. NetApp helps organizations meet the complexities created by rapid data and cloud growth, multi-cloud management, and the adoption of next-generation technologies, such as AI, Kubernetes, and modern databases. Our modern approach to hybrid multicloud infrastructure and data management, which we
```

**Now, we use neo4j as a vector DB, with the help of Langchain, we convert it as a retriever. Using OpenAI for chat.**

**And another chain for OpenAI chat again provided by Langchain:**

```
neo4j_vector_store = Neo4jVector.from_existing_graph(
    embedding=OpenAIEmbeddings(),
    url=NEO4J_URI,
    username=NEO4J_USERNAME,
    password=NEO4J_PASSWORD,
    index_name=VECTOR_INDEX_NAME,
    node_label=VECTOR_NODE_LABEL,
    text_node_properties=[VECTOR_SOURCE_PROPERTY],
    embedding_node_property=VECTOR_EMBEDDING_PROPERTY,
)

retriever = neo4j_vector_store.as_retriever()

chain = RetrievalQAWithSourcesChain.from_chain_type(
    ChatOpenAI(temperature=0),
    chain_type="stuff",
    retriever=retriever
)

def prettychain(question: str) -> str:
    """Pretty print the chain's response to a question"""
    response = chain({"question": question},
                     return_only_outputs=True)
    print(textwrap.fill(response['answer'], 80))
```

## Final:

```
question = "What is Netapp's primary business?"

prettychain(question)

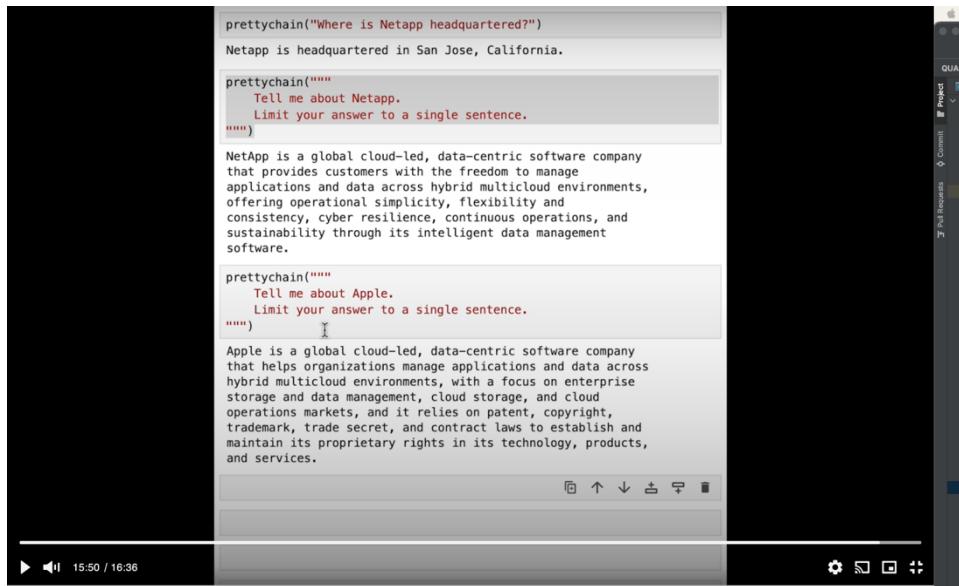
NetApp's primary business is enterprise storage and data management, cloud storage, and cloud operations. They focus on providing solutions for hybrid multicloud environments and helping organizations manage applications and data across these environments. They also offer a portfolio of cloud services and storage infrastructure powered by intelligent data management software. Their products and services are used by a diverse customer base in various industries such as energy, financial services, government, technology, healthcare, manufacturing, media, and telecommunications.

prettychain("Where is Netapp headquartered?")
Netapp is headquartered in San Jose, California.

prettychain("""
    Tell me about Netapp.
    Limit your answer to a single sentence.
""")

NetApp is a global cloud-led, data-centric software company that provides customers with the freedom to manage applications and data across hybrid multicloud environments, offering operational simplicity, flexibility and consistency, cyber resilience, continuous operations, and sustainability through its intelligent data management software.
```

# Example of hallucination, even for apple the answer was similar to netapp.



```
prettychain("Where is Netapp headquartered?")
Netapp is headquartered in San Jose, California.

prettychain(""""
    Tell me about Netapp.
    Limit your answer to a single sentence.
""")  
NetApp is a global cloud-led, data-centric software company that provides customers with the freedom to manage applications and data across hybrid multicloud environments, offering operational simplicity, flexibility and consistency, cyber resilience, continuous operations, and sustainability through its intelligent data management software.

prettychain(""""
    Tell me about Apple.
    Limit your answer to a single sentence.
""")  
Apple is a global cloud-led, data-centric software company that helps organizations manage applications and data across hybrid multicloud environments, with a focus on enterprise storage and data management, cloud storage, and cloud operations markets, and it relies on patent, copyright, trademark, trade secret, and contract laws to establish and maintain its proprietary rights in its technology, products, and services.
```

15:50 / 16:36

## Simple prompt engineering:



```
prettychain(""""
    Tell me about Apple.
    Limit your answer to a single sentence.
    If you are unsure about the answer, say you don't know.
""")  
I don't know about Apple.
```

16:20 / 16:36

**Note: in this chapter, we used neo4j as a vector store, not as a Kg. Also there was no relationship between these chunks.**