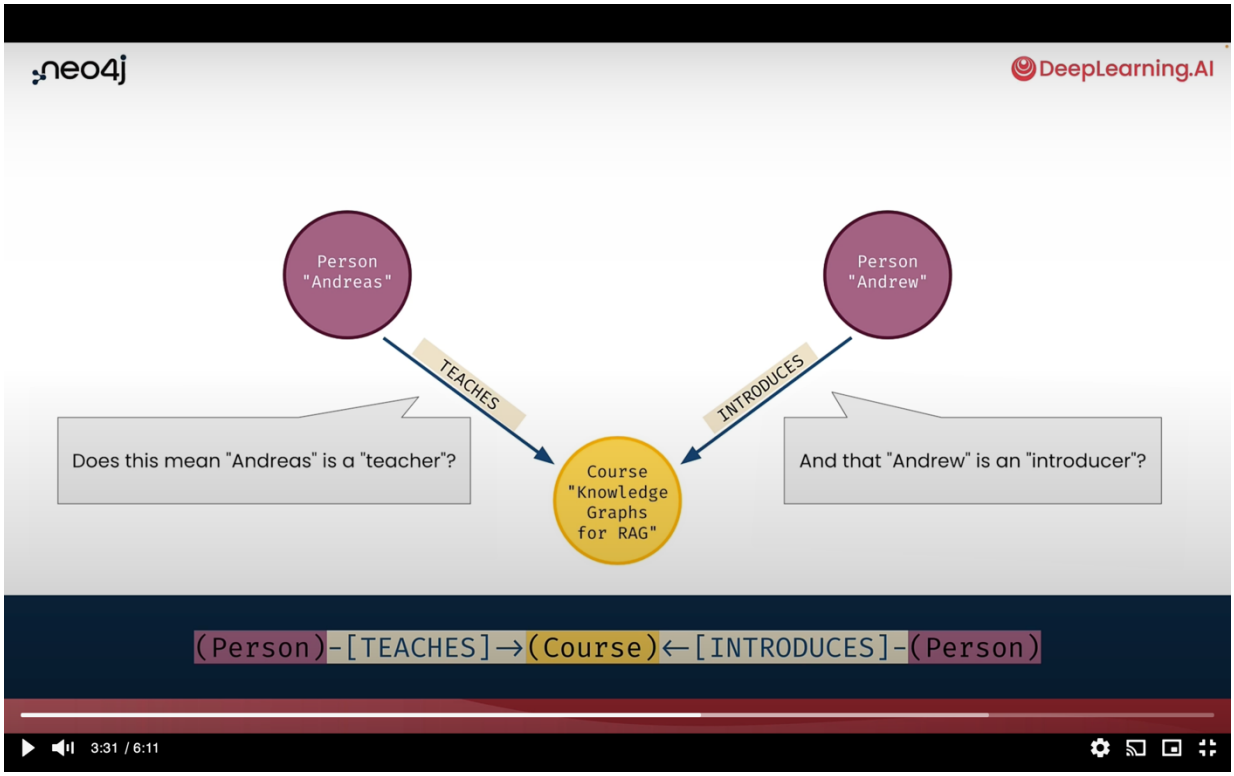
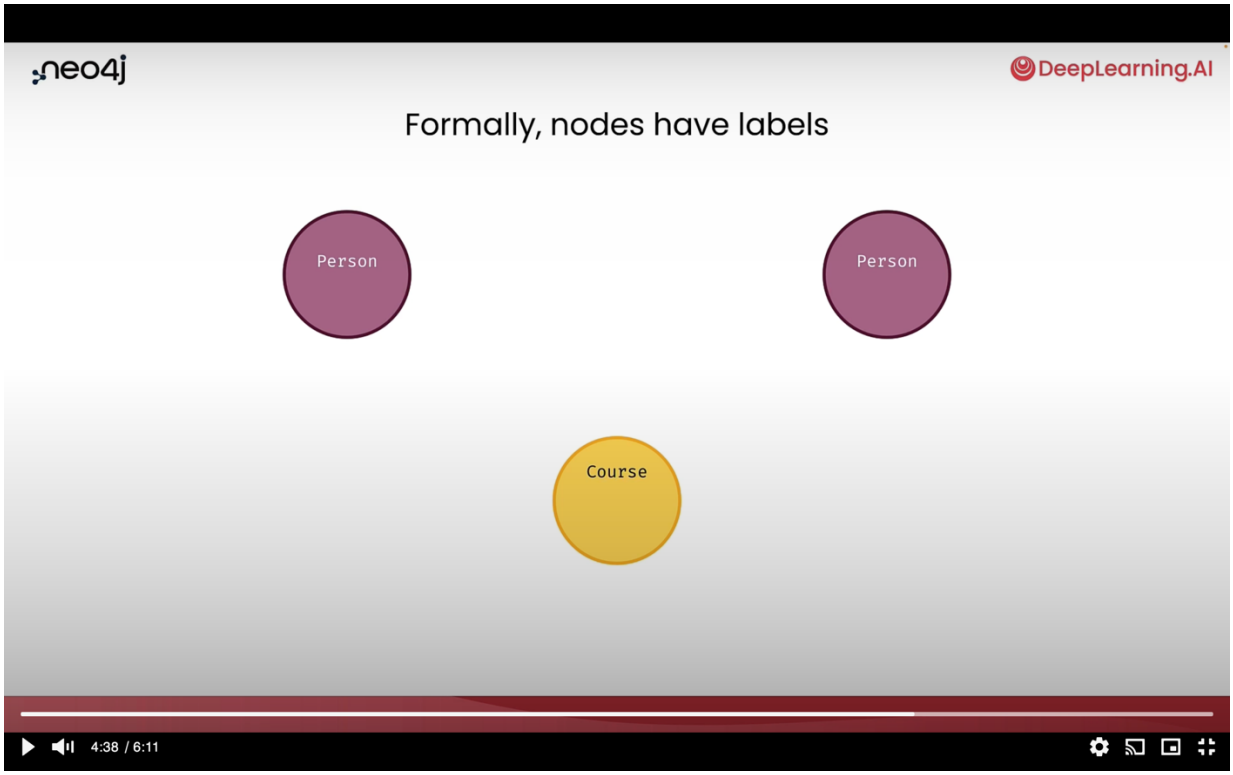
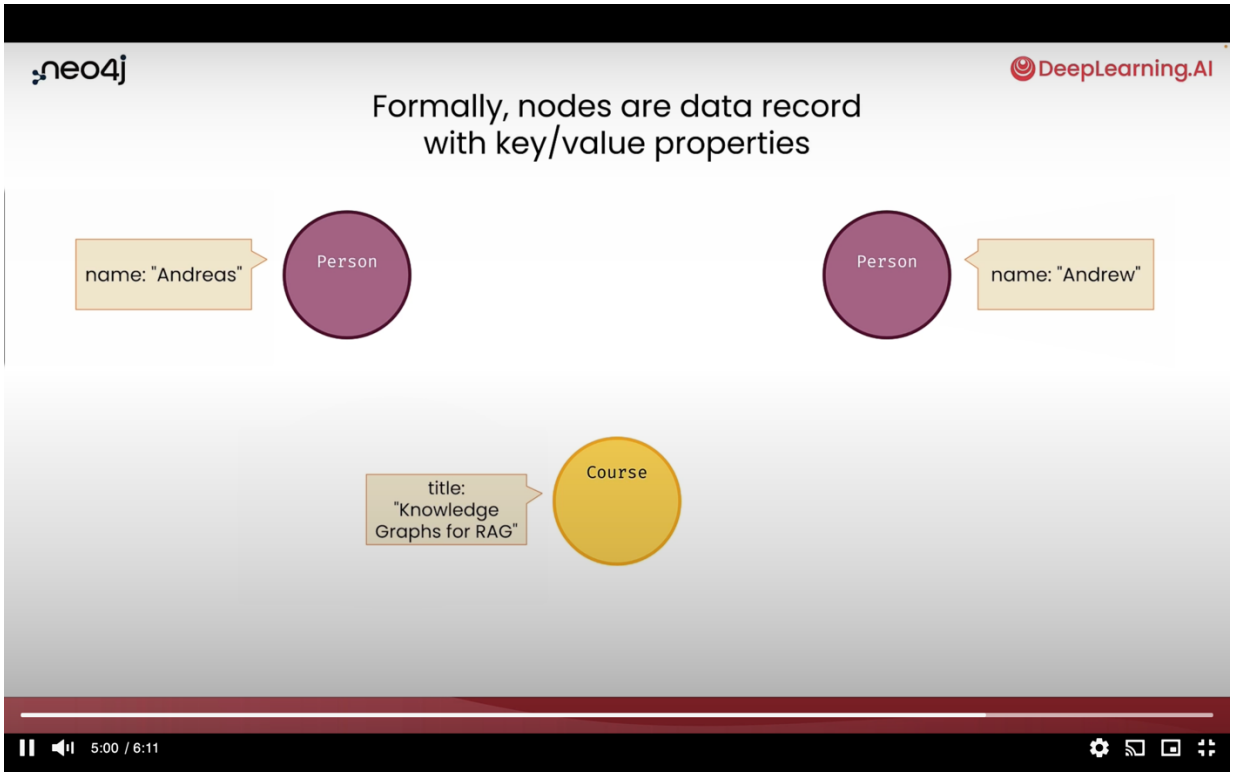
Course: https://www.deeplearning.ai/short-courses/knowledge-graphs-rag/

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

KG fundamentals:



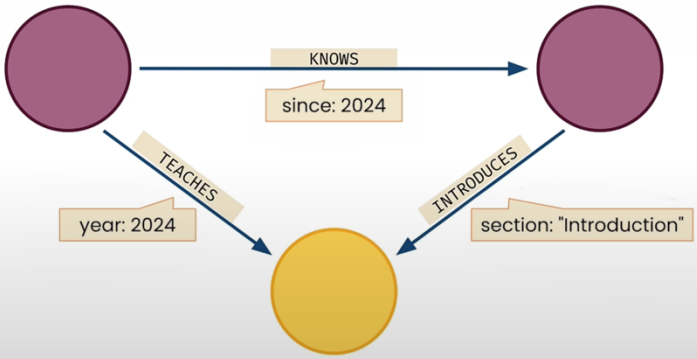

neo4j
DeepLearning.AI
Formally, nodes have labels
Person
Person
Course
4:38 / 6:11


neo4j
DeepLearning.AI
Formally, nodes are data record
with key/value properties
name: "Andreas"
Person
Person
name: "Andrew"
title:
"Knowledge
Graphs for RAG"
Course
5:00 / 6:11

neo4j

DeepLearning.AI

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



5:27 / 6:11

neo4j

DeepLearning.AI

What is a Knowledge Graph?

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

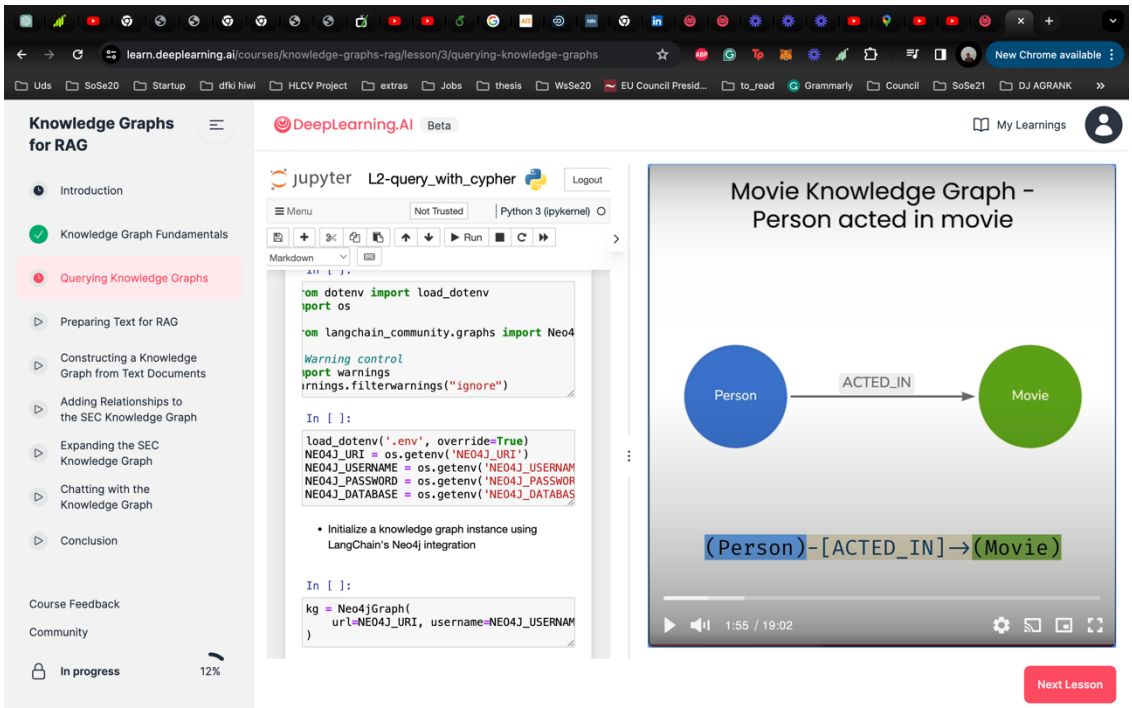
Both nodes and relationships can have **properties**.

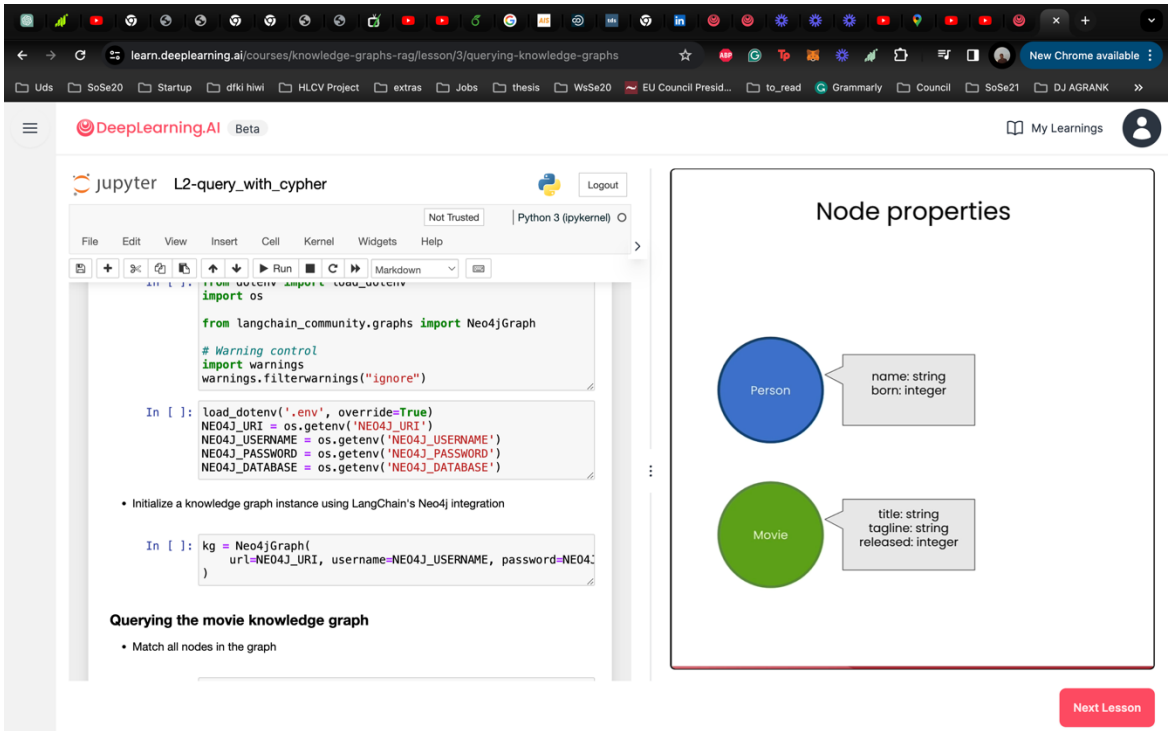
Nodes can be given **labels** to group them together.

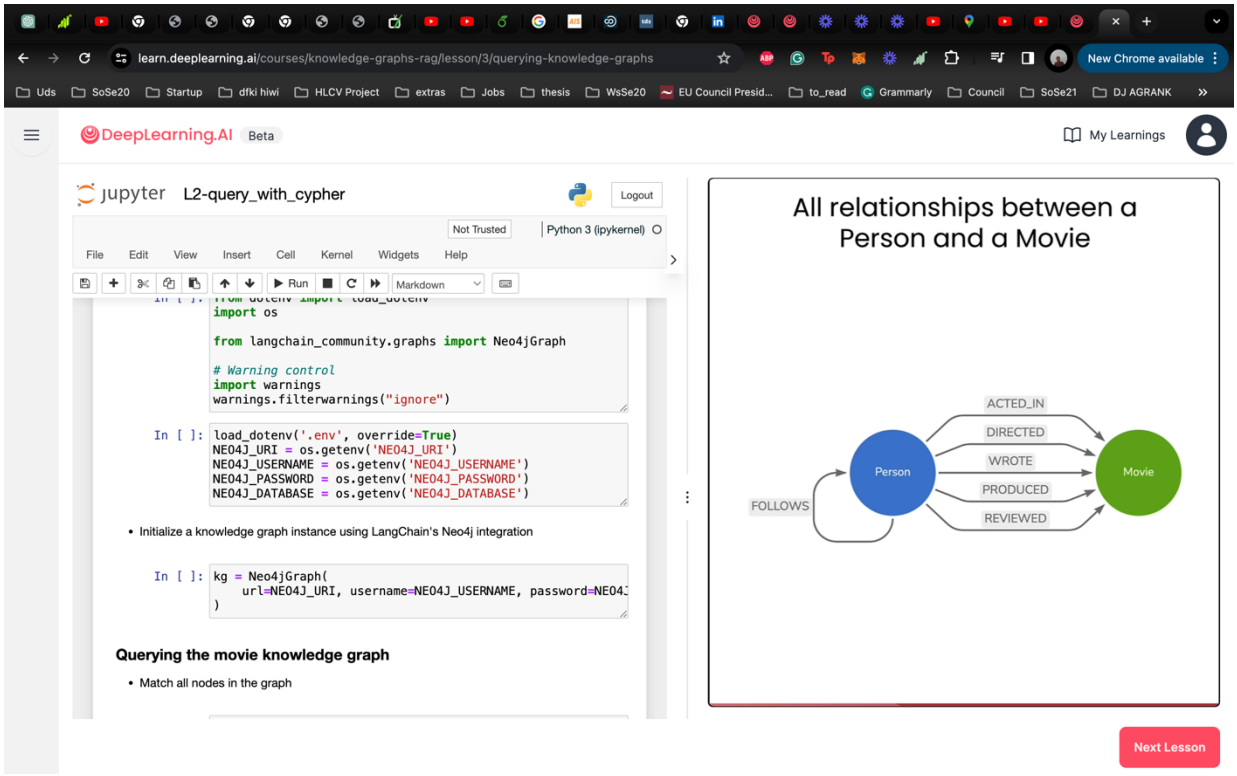
Relationships always have a **type** and a **direction**.

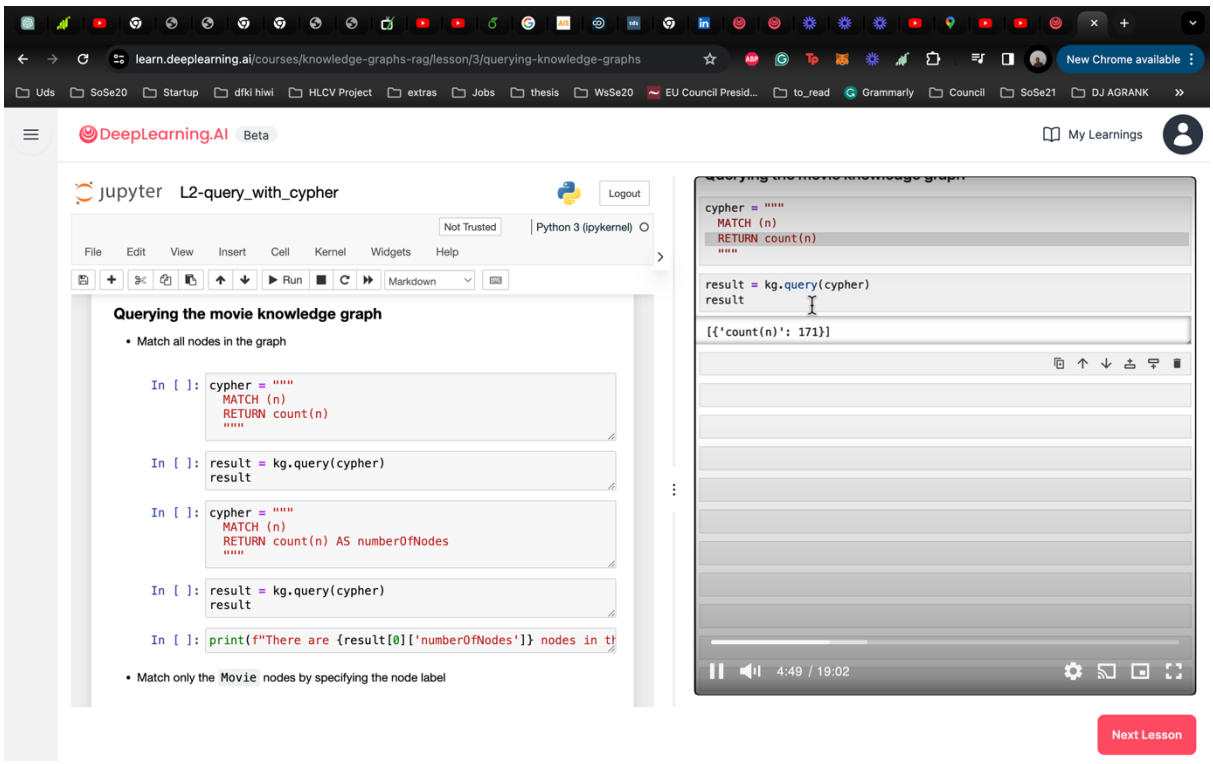
5:57 / 6:11

Query KG:


learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
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%
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
Run
Markdown
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAM
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWOR
NEO4J_DATABASE = os.getenv('NEO4J_DATABAS
• Initialize a knowledge graph instance using LangChain's Neo4j integration
kg = Neo4jGraph(
url=NEO4J_URI, username=NEO4J_USERNAM
)
Movie Knowledge Graph - Person acted in movie
Person
ACTED_IN
Movie
(Person)-[ACTED_IN]→(Movie)
1:55 / 19:02
Next Lesson

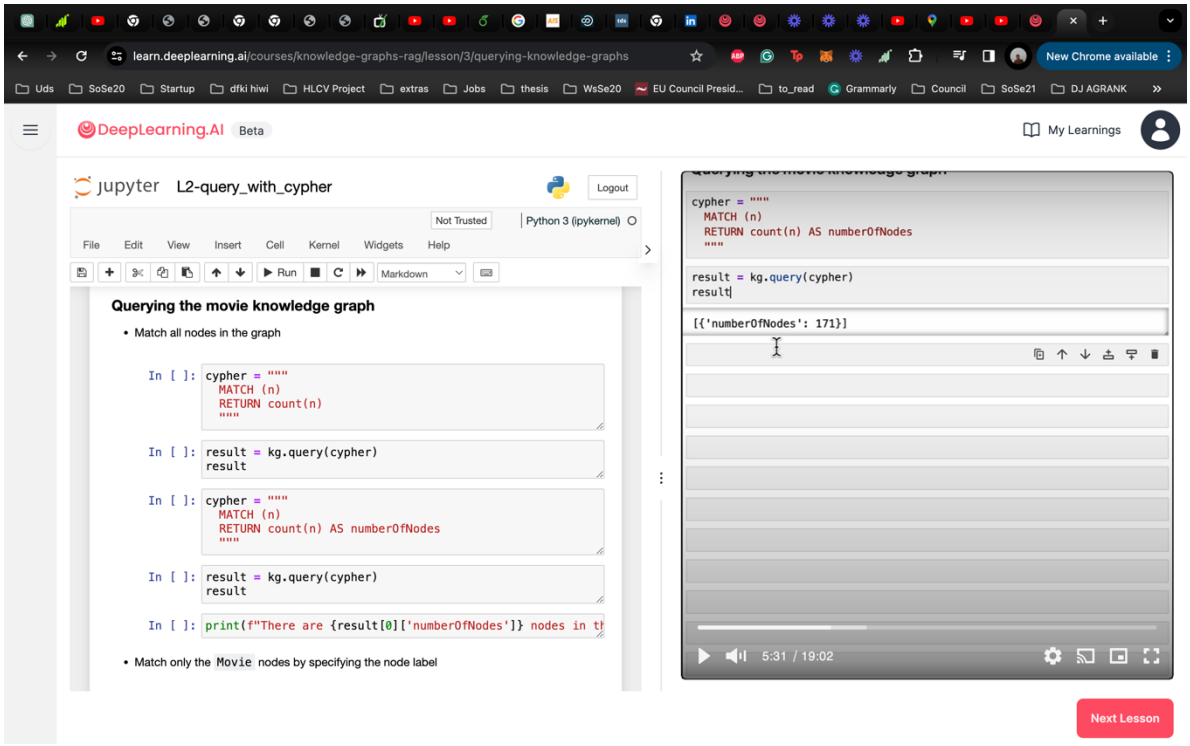

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
import os
from langchain_community.graphs import Neo4jGraph
# Warning control
import warnings
warnings.filterwarnings("ignore")
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
kg = Neo4jGraph(
url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J
)
Querying the movie knowledge graph
• Match all nodes in the graph
Node properties
Person
name: string
born: integer
Movie
title: string
tagline: string
released: integer
Next Lesson



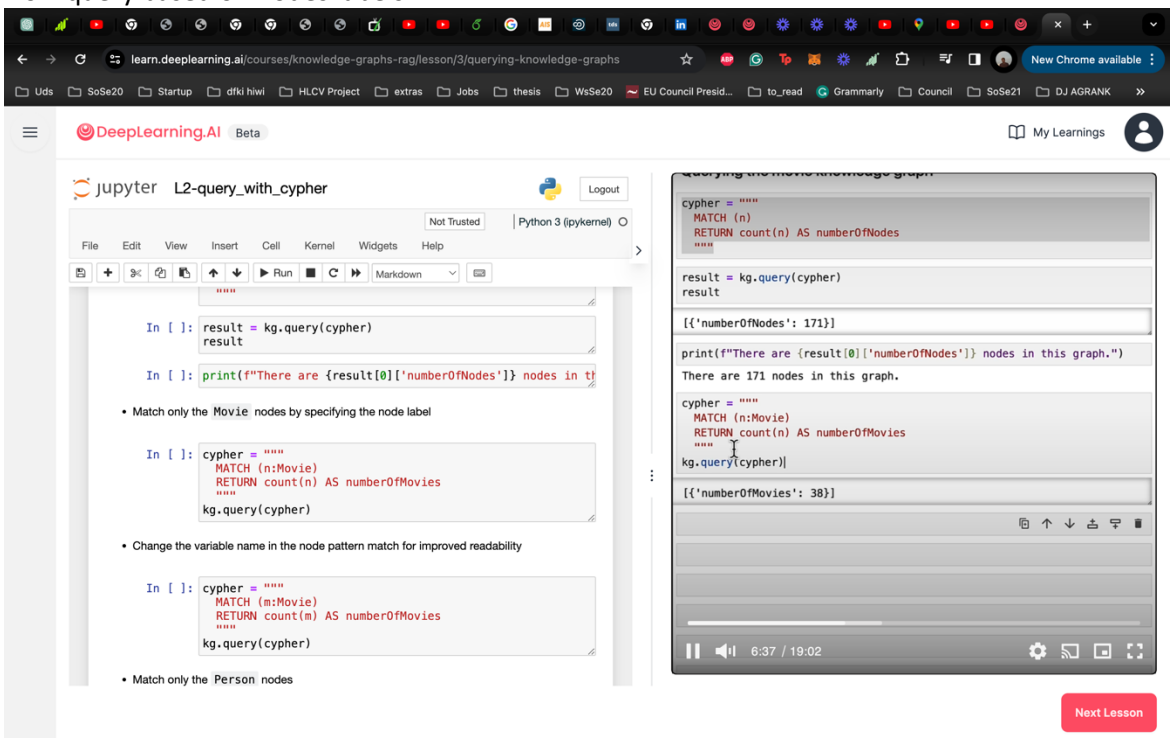


Total 171 nodes in the graph.
Return is a dict, where key is what we ask in query.
We use cypher as query language for neo4j.

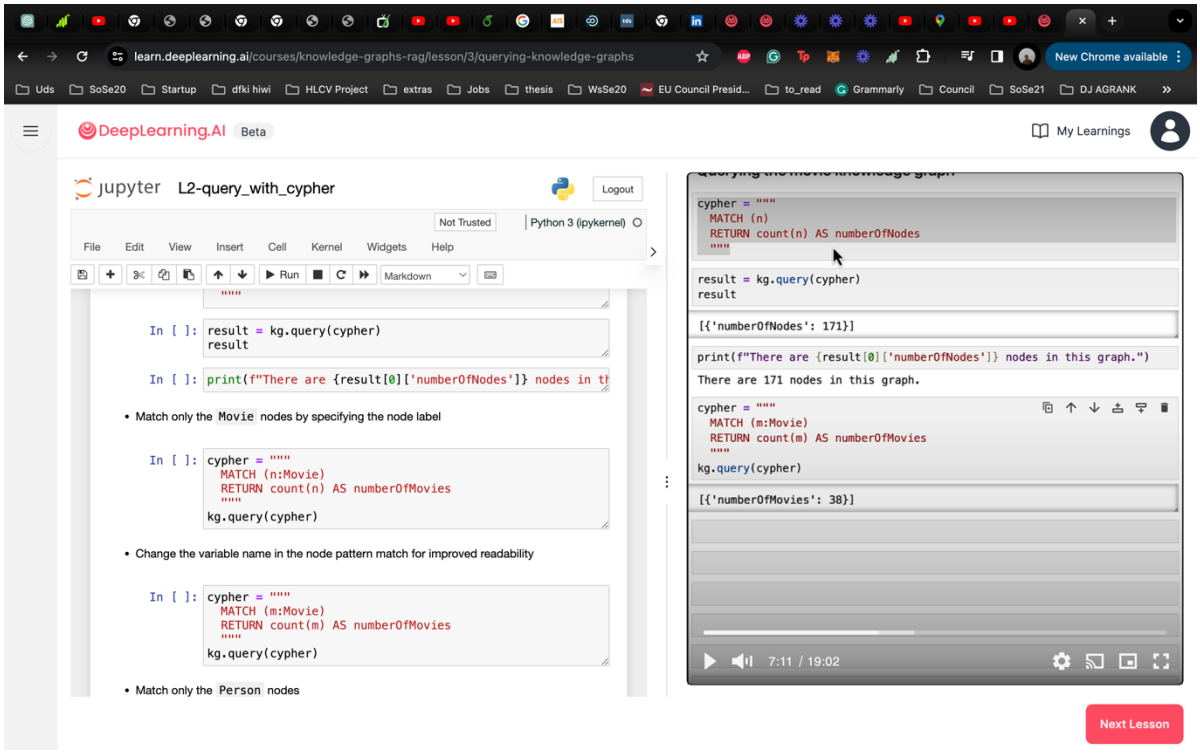
We can rename the key:

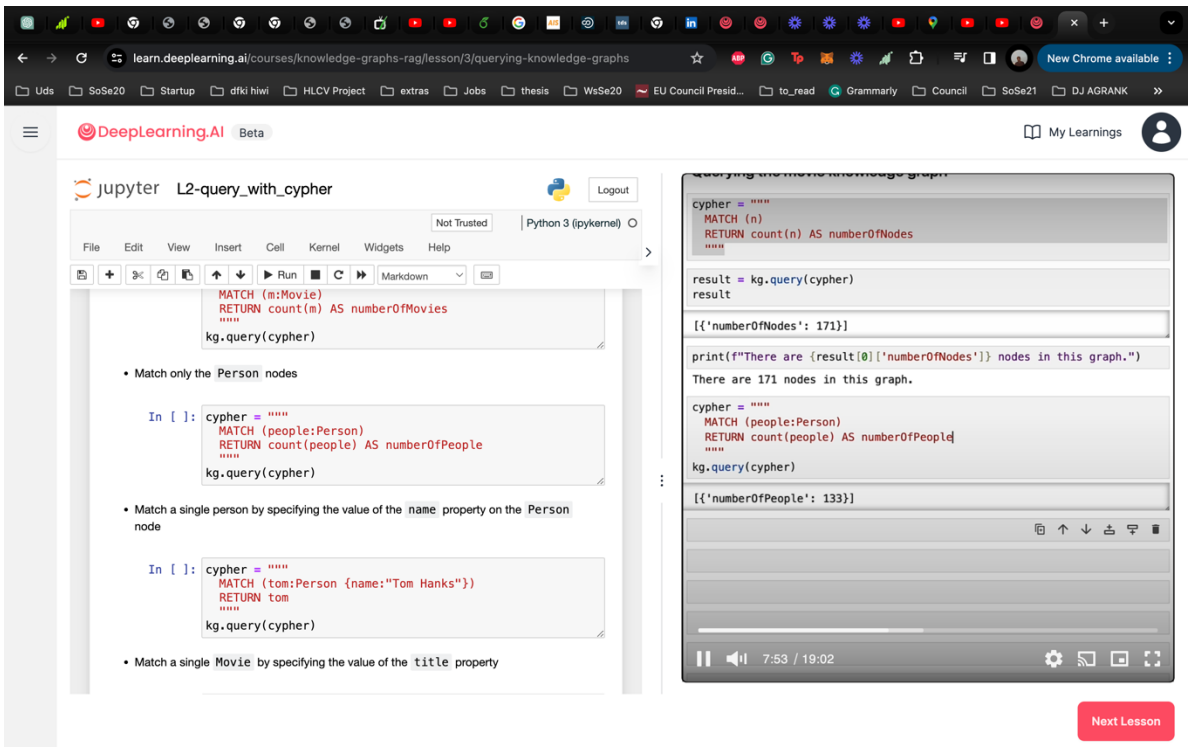


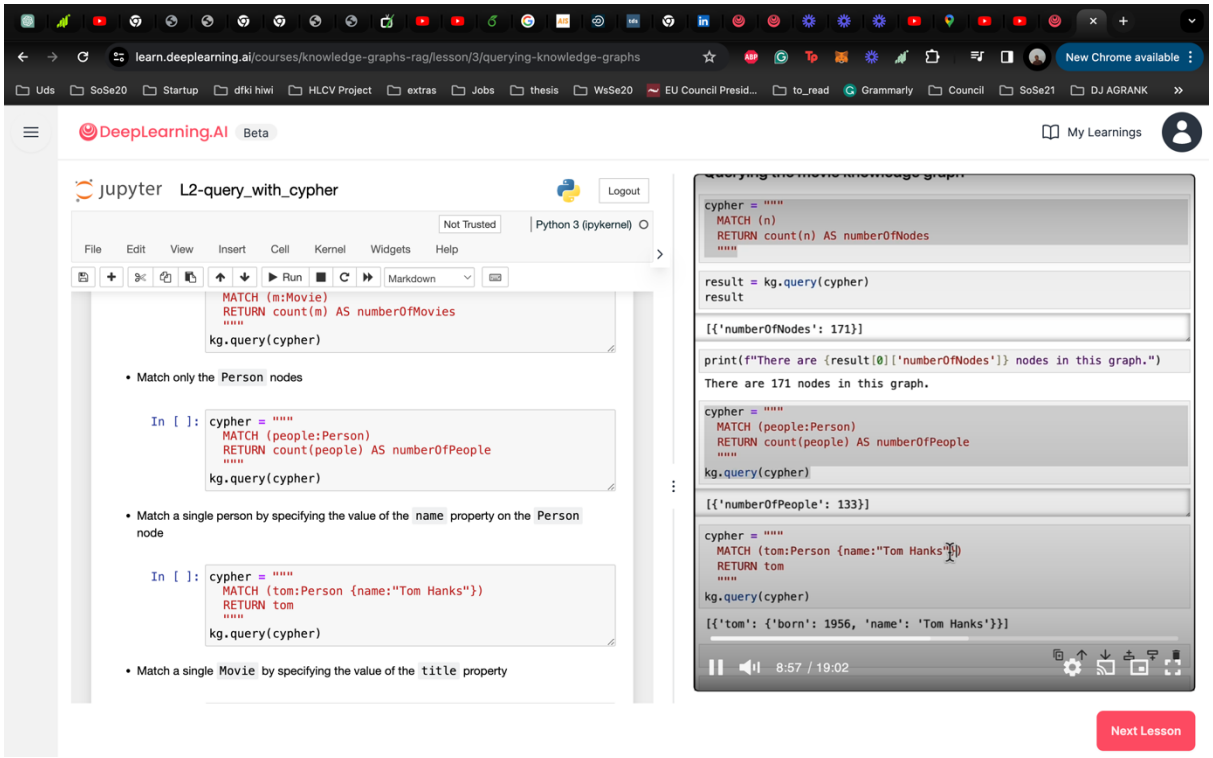
Now query based on nodes labels.

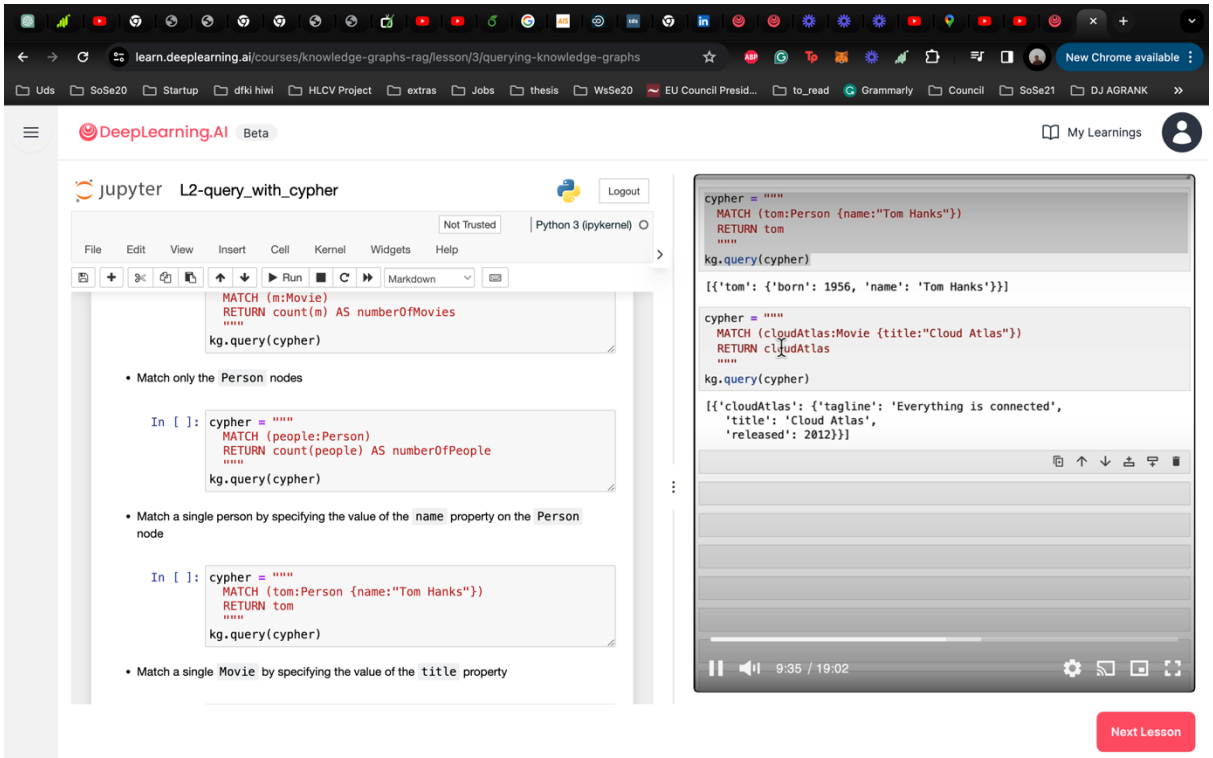


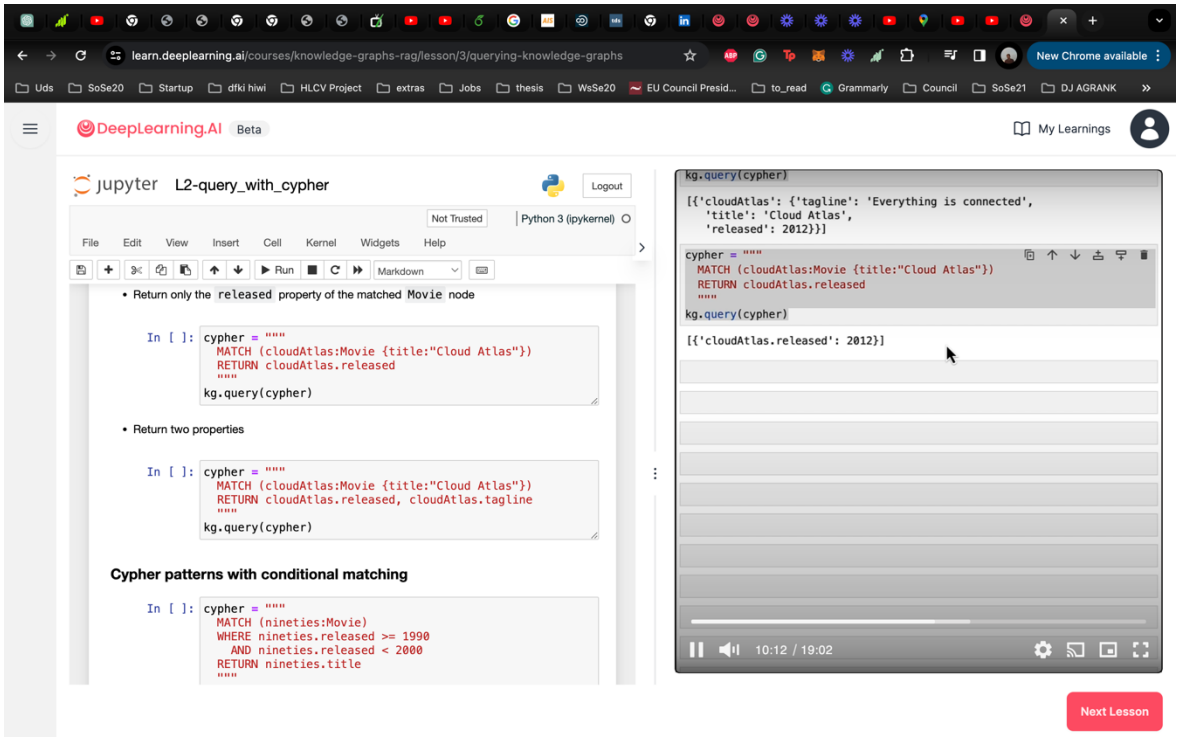
Can change name of variable ,n to m

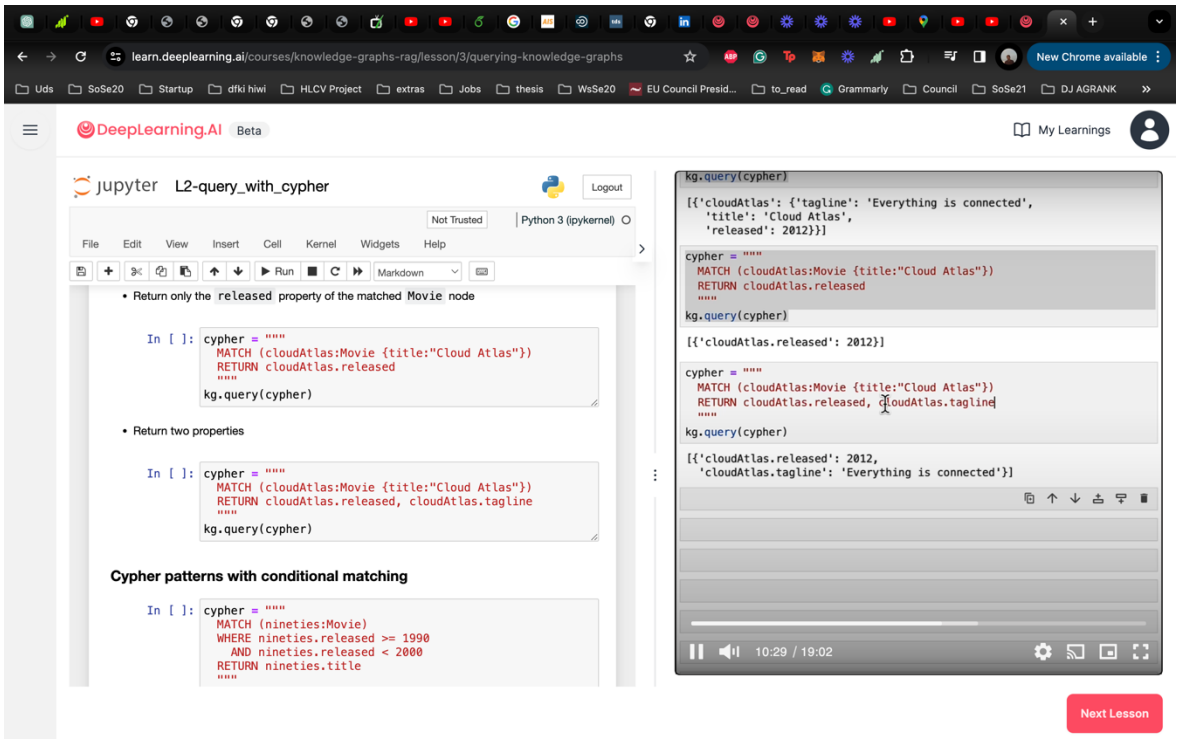

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
In [ ]:
result = kg.query(cypher)
result
In [ ]:
print(f"There are {result[0]['numberOfNodes']} nodes in th
• 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
cypher = """
MATCH (n)
RETURN count(n) AS numberOfNodes
"""
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 (m:Movie)
RETURN count(m) AS numberOfMovies
"""
kg.query(cypher)
[{'numberOfMovies': 38}]
7:11 / 19:02
Next Lesson


learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
MATCH (m: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
cypher = """
MATCH (n)
RETURN count(n) AS numberOfNodes
"""
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 numberOfPeople
"""
kg.query(cypher)
[{'numberOfPeople': 133}]
7:53 / 19:02
Next Lesson

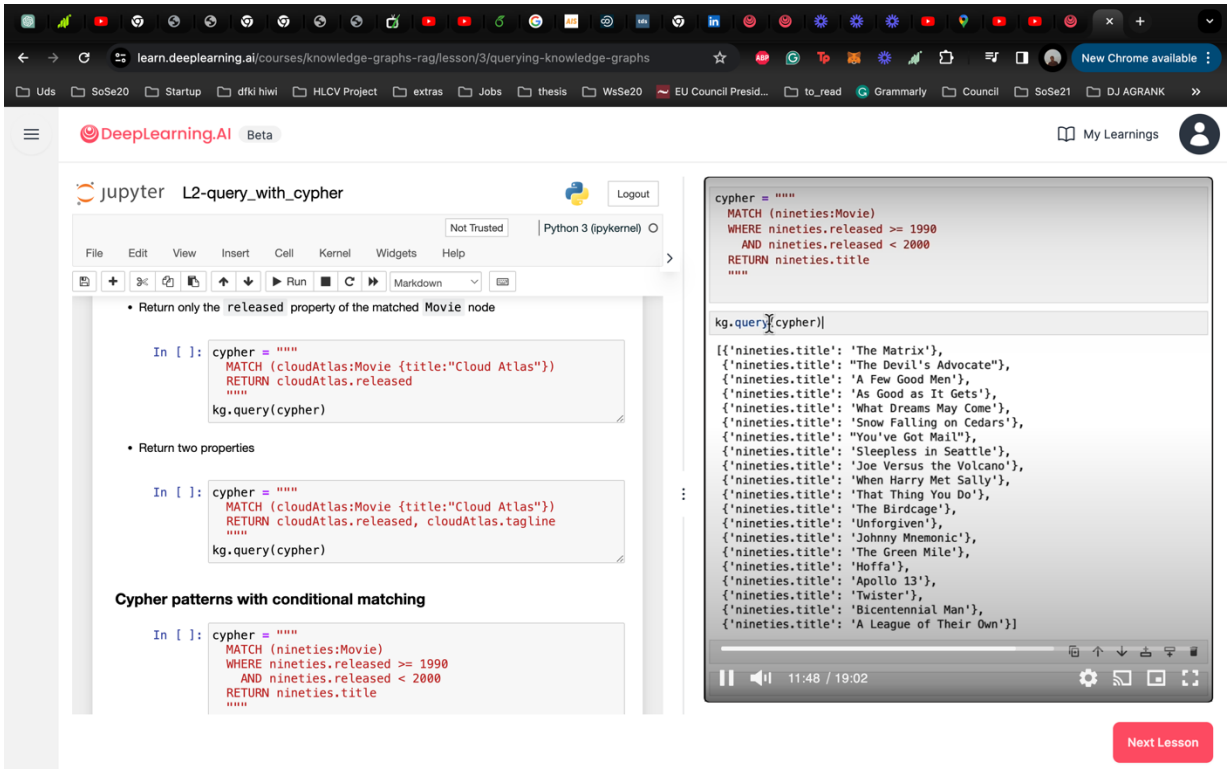

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
MATCH (m:Movie)
RETURN count(m) 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
cypher = """
MATCH (n)
RETURN count(n) AS numberOfNodes
"""
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 numberOfPeople
"""
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
Next Lesson

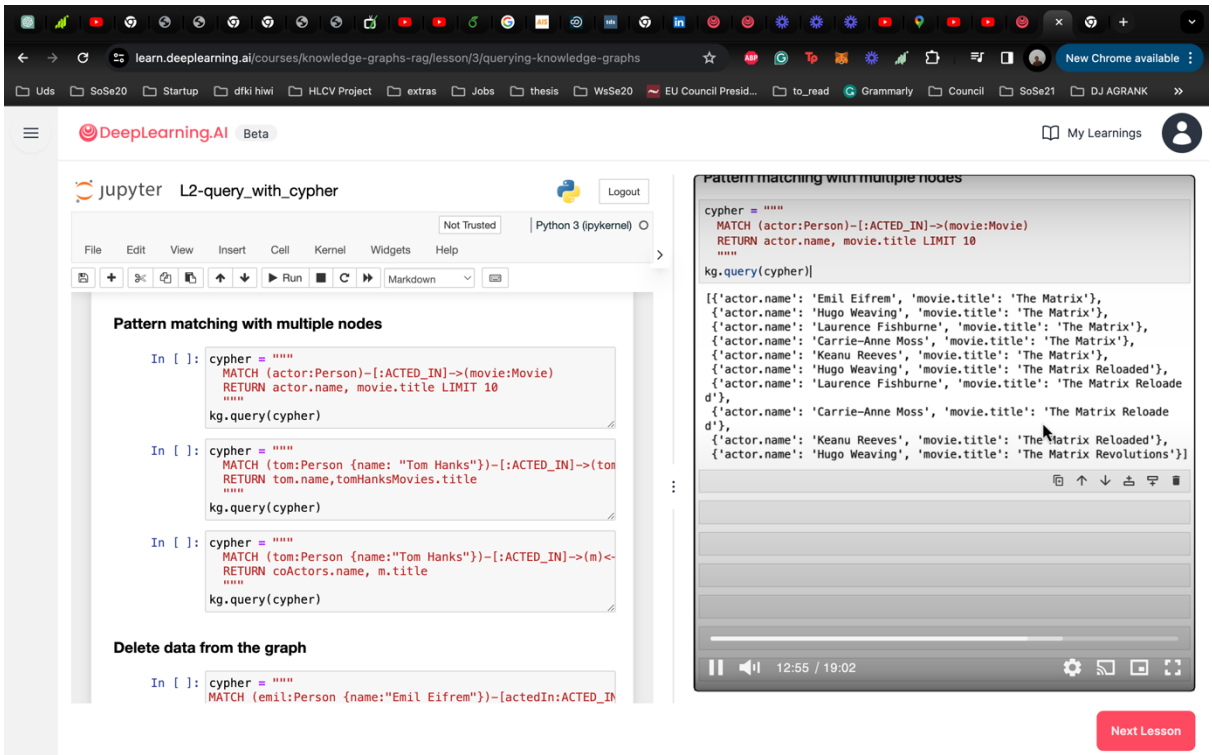

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
MATCH (m:Movie)
RETURN count(m) 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
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
Next Lesson

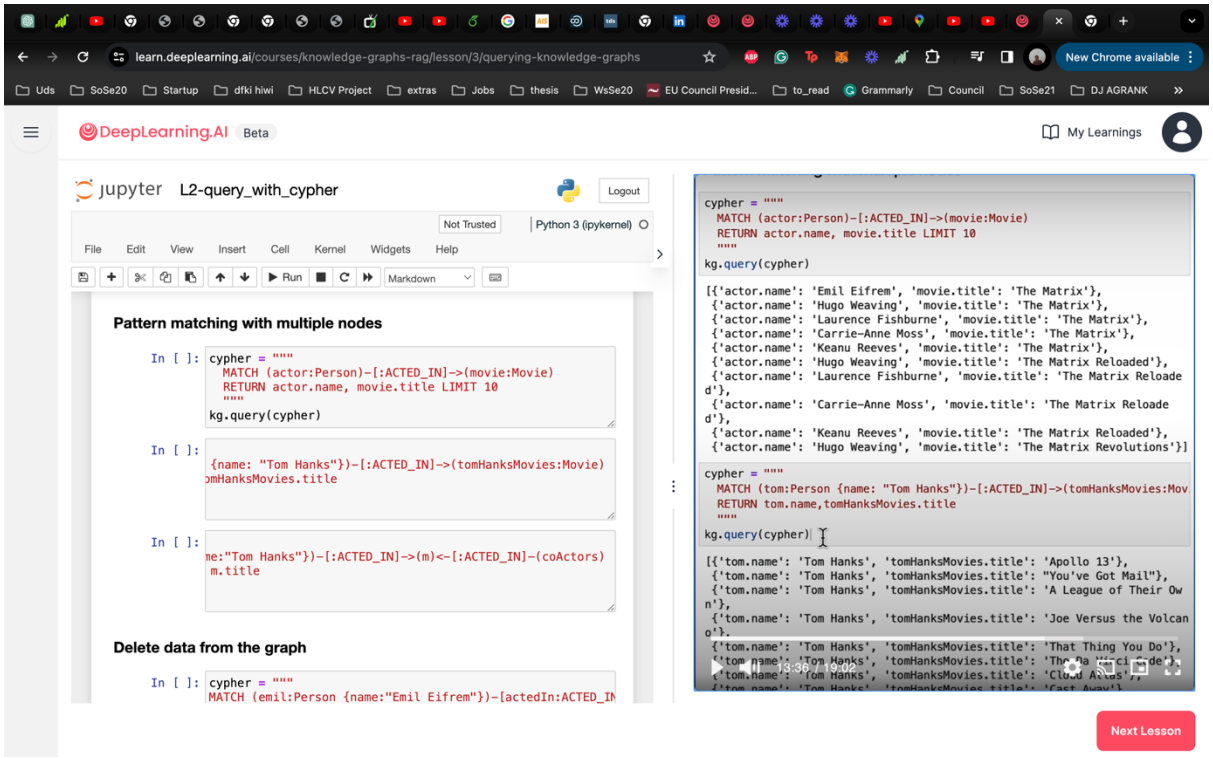

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
• Return only the released property of the matched Movie node
In [ ]:
cypher = """
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})
RETURN cloudAtlas.released
"""
kg.query(cypher)
• Return two properties
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
"""
kg.query(cypher)
[{'cloudAtlas': {'tagline': 'Everything is connected',
'title': 'Cloud Atlas',
'released': 2012}}]
[{'cloudAtlas.released': 2012}]
10:12 / 19:02
Next Lesson

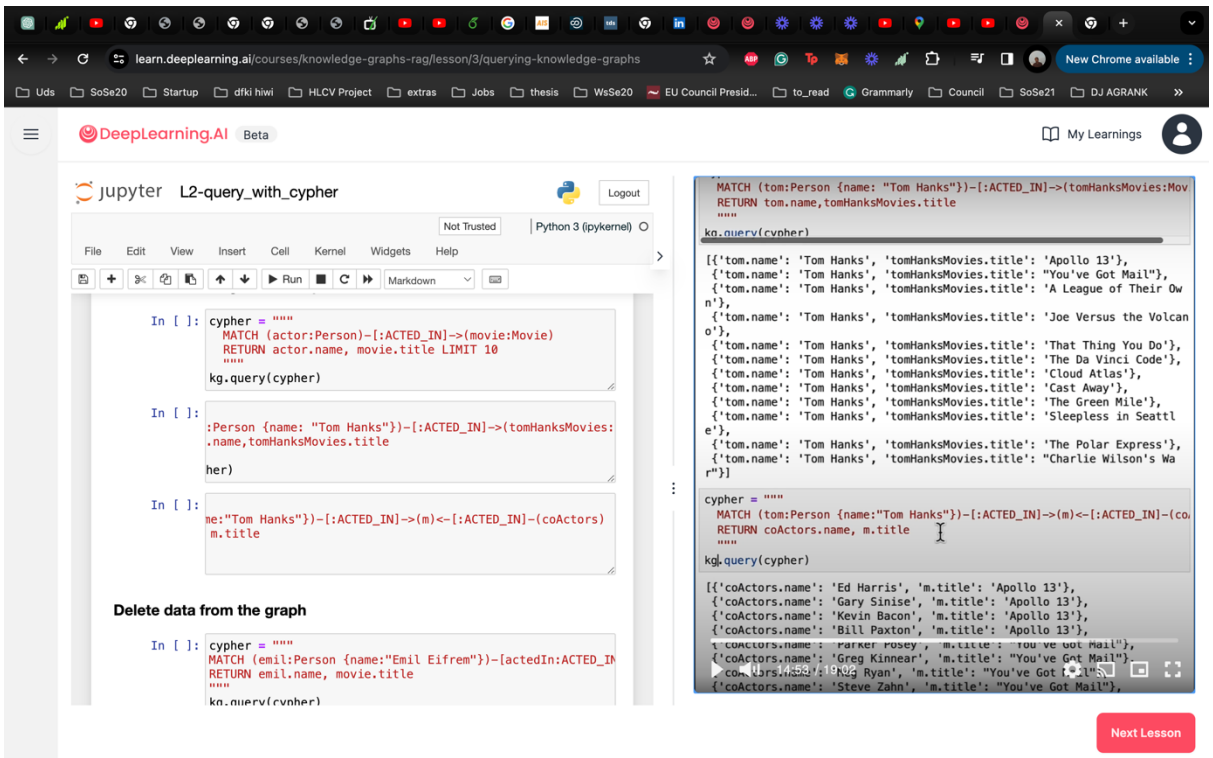

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
• Return only the released property of the matched Movie node
In [ ]:
cypher = """
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})
RETURN cloudAtlas.released
"""
kg.query(cypher)
• Return two properties
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
"""
kg.query(cypher)
[{'cloudAtlas': {'tagline': 'Everything is connected',
'title': 'Cloud Atlas',
'released': 2012}}]
[{'cloudAtlas.released': 2012}]
[{'cloudAtlas.released': 2012,
'cloudAtlas.tagline': 'Everything is connected'}]
10:29 / 19:02
Next Lesson

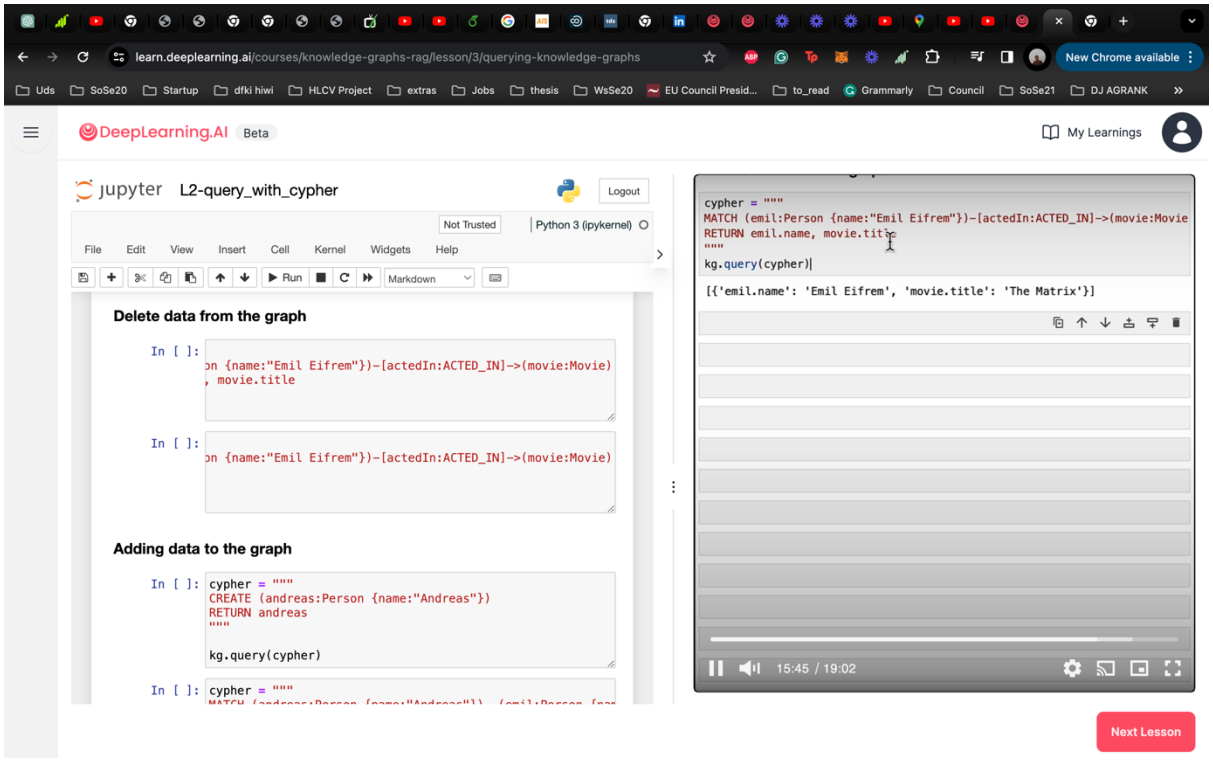
Query range:


learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI Beta
My Learnings
jupyter L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
• Return only the released property of the matched Movie node
In [ ]:
cypher = """
MATCH (cloudAtlas:Movie {title:"Cloud Atlas"})
RETURN cloudAtlas.released
"""
kg.query(cypher)
• Return two properties
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
"""
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

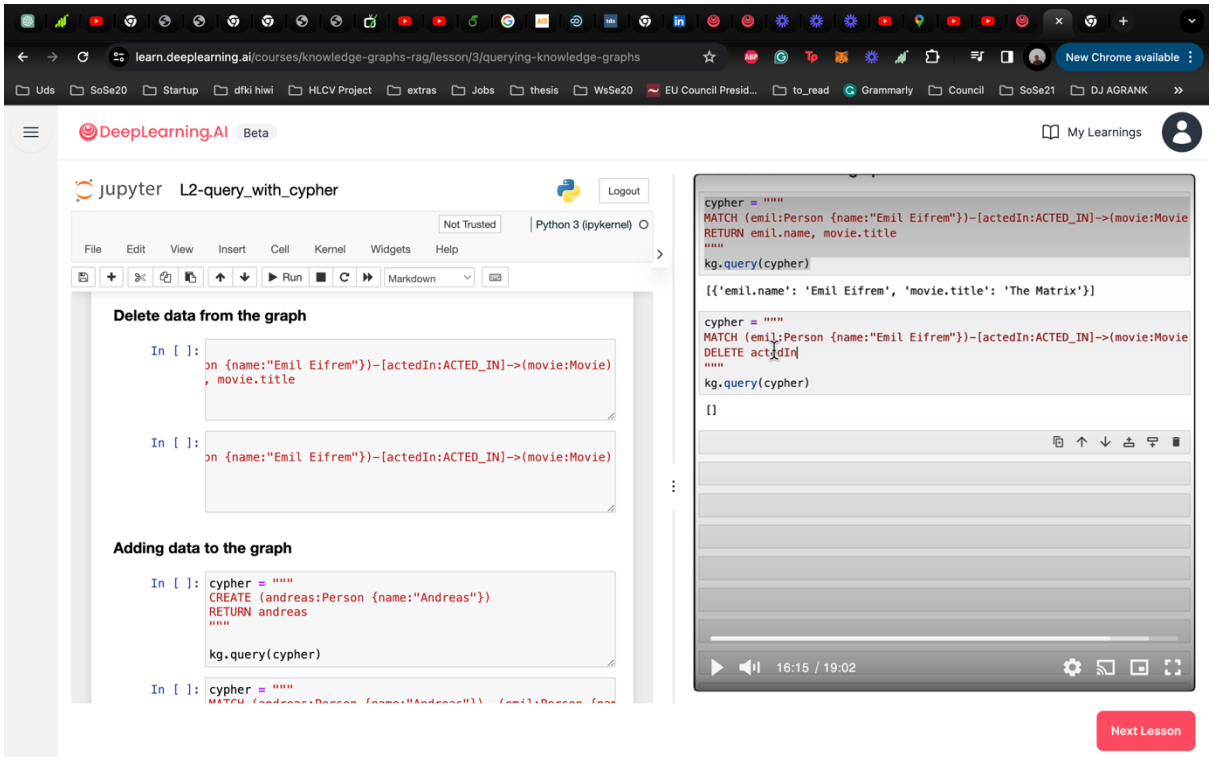

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI Beta
My Learnings
jupyter L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
Pattern matching with multiple nodes
In [ ]:
cypher = """
MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)
RETURN actor.name, movie.title LIMIT 10
"""
kg.query(cypher)
In [ ]:
cypher = """
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tom
RETURN tom.name,tomHanksMovies.title
"""
kg.query(cypher)
In [ ]:
cypher = """
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-
RETURN coActors.name, m.title
"""
kg.query(cypher)
Delete data from the graph
In [ ]:
cypher = """
MATCH (emil:Person {name:"Emil Eifrem"})-[actedIn:ACTED_IN
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 Reloade
d'},
{'actor.name': 'Carrie-Anne Moss', 'movie.title': 'The Matrix Reloade
d'},
{'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
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
Pattern matching with multiple nodes
In [ ]:
cypher = """
MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)
RETURN actor.name, movie.title LIMIT 10
"""
kg.query(cypher)
Delete data from the graph
Next Lesson

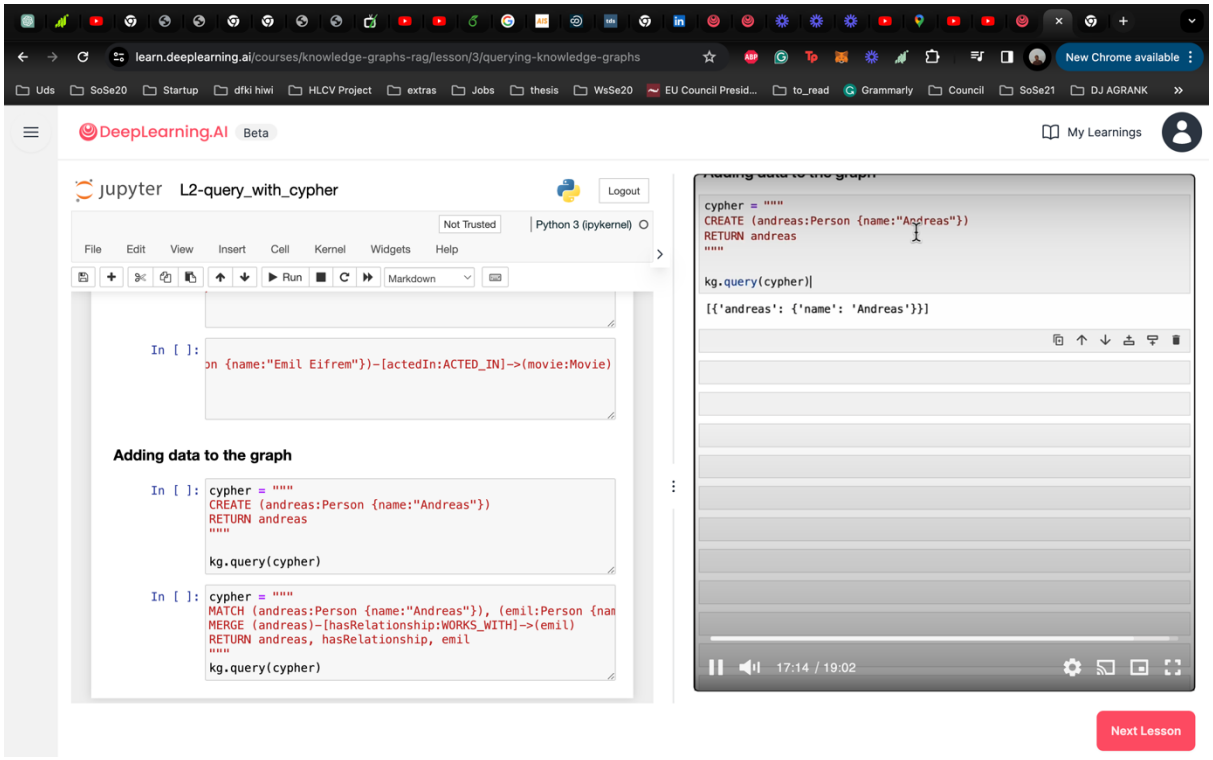

learn.deeplearning.ai/courses/knowledge-graphs-rag/lesson/3/querying-knowledge-graphs
New Chrome available
DeepLearning.AI
Beta
My Learnings
jupyter
L2-query_with_cypher
Logout
Not Trusted
Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
Delete data from the graph
Next Lesson

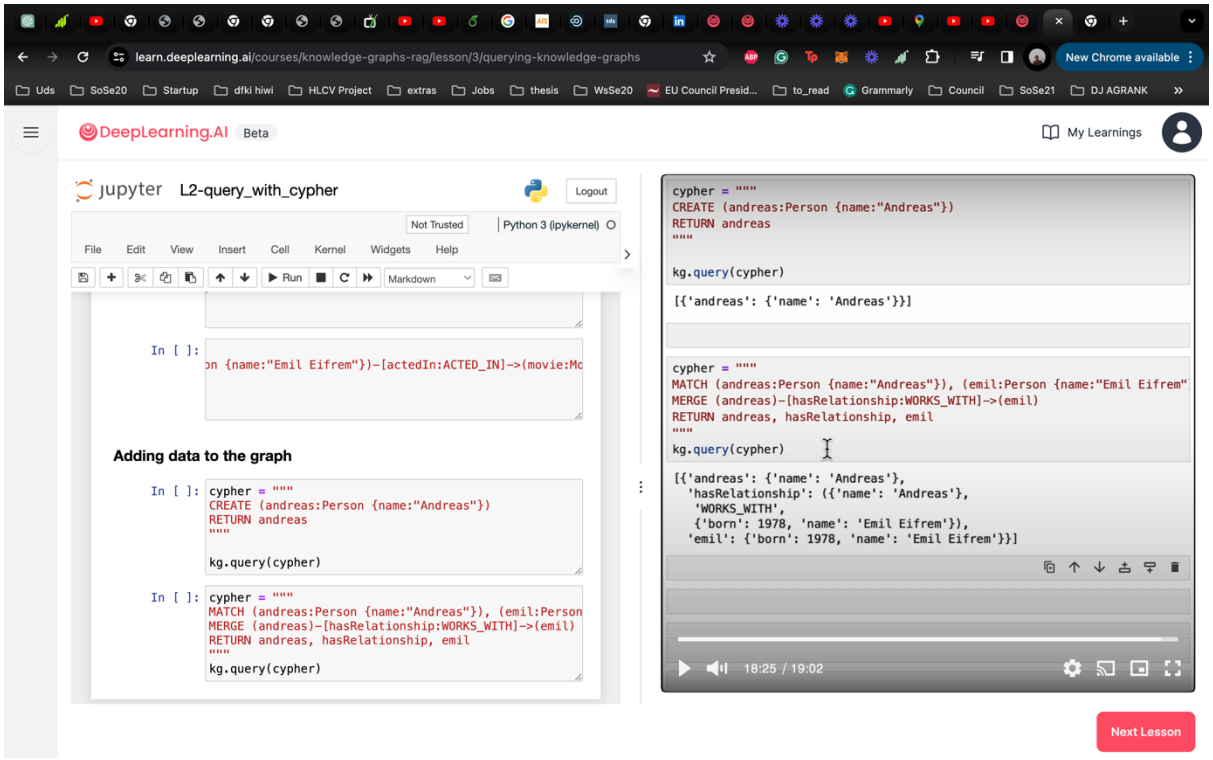


Delete:



Adding:

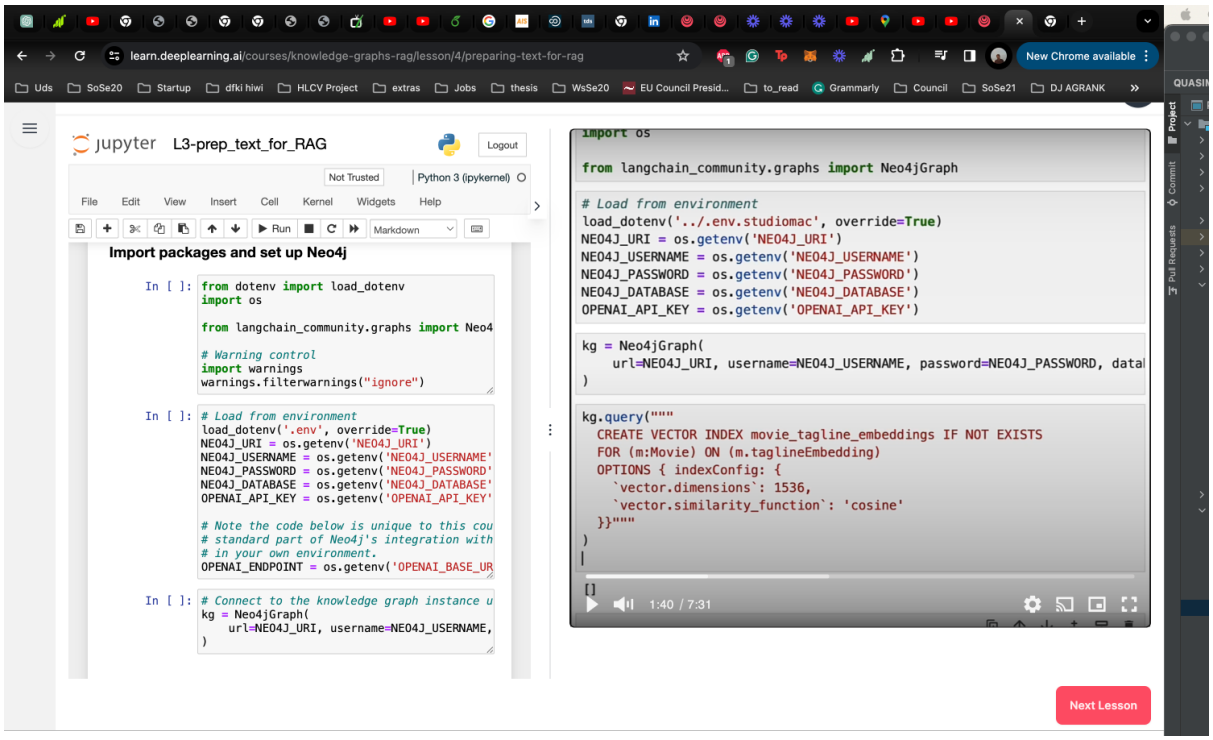




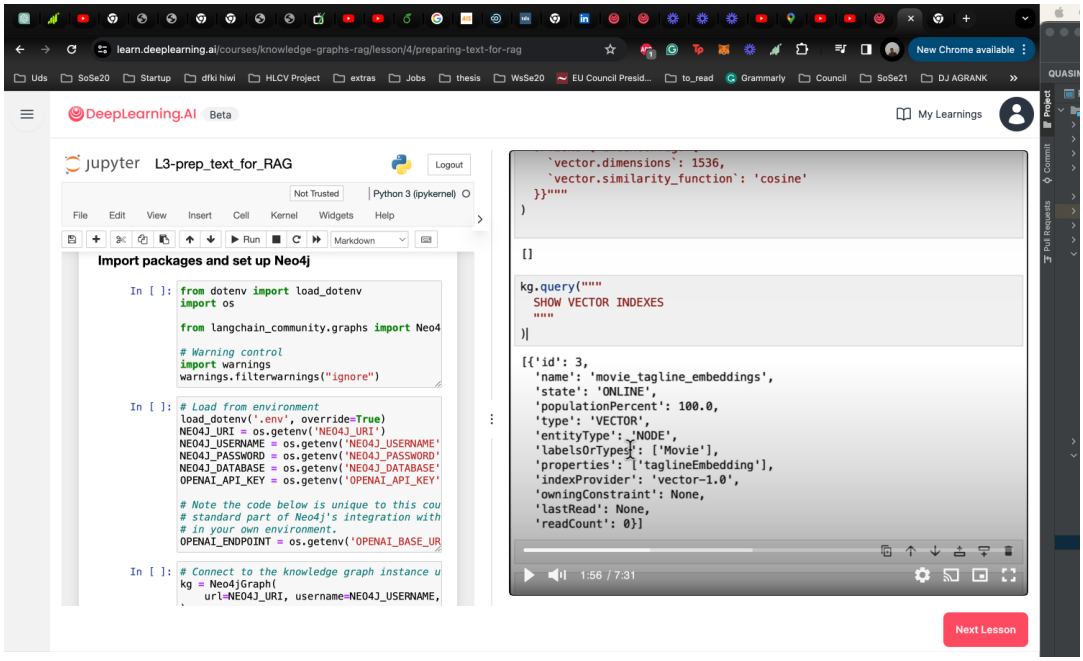
Here, we first find individual persons, without any relationship, hence comma in match.
Then we use MERGE, merge is similar to create but it the relationship already exist, it wont create it again. Hence merge.

NEXT CHAPTER:

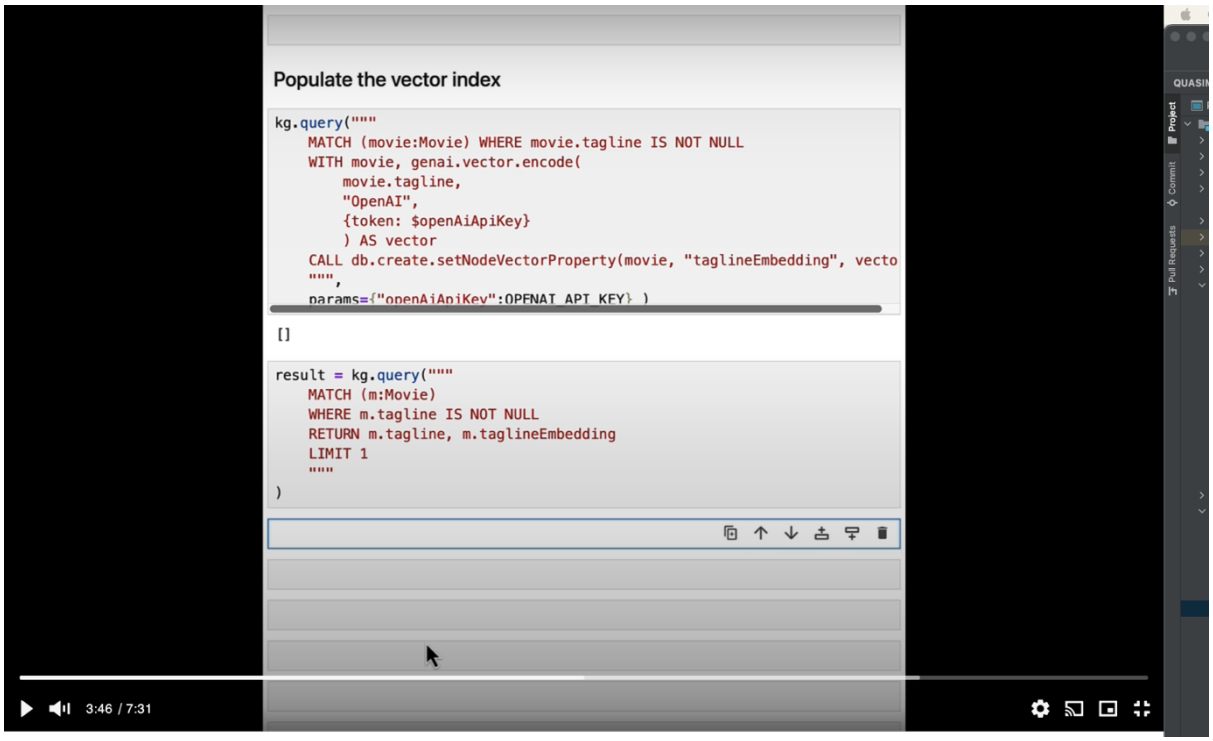
Lesson 3: Preparing Text Data for RAG



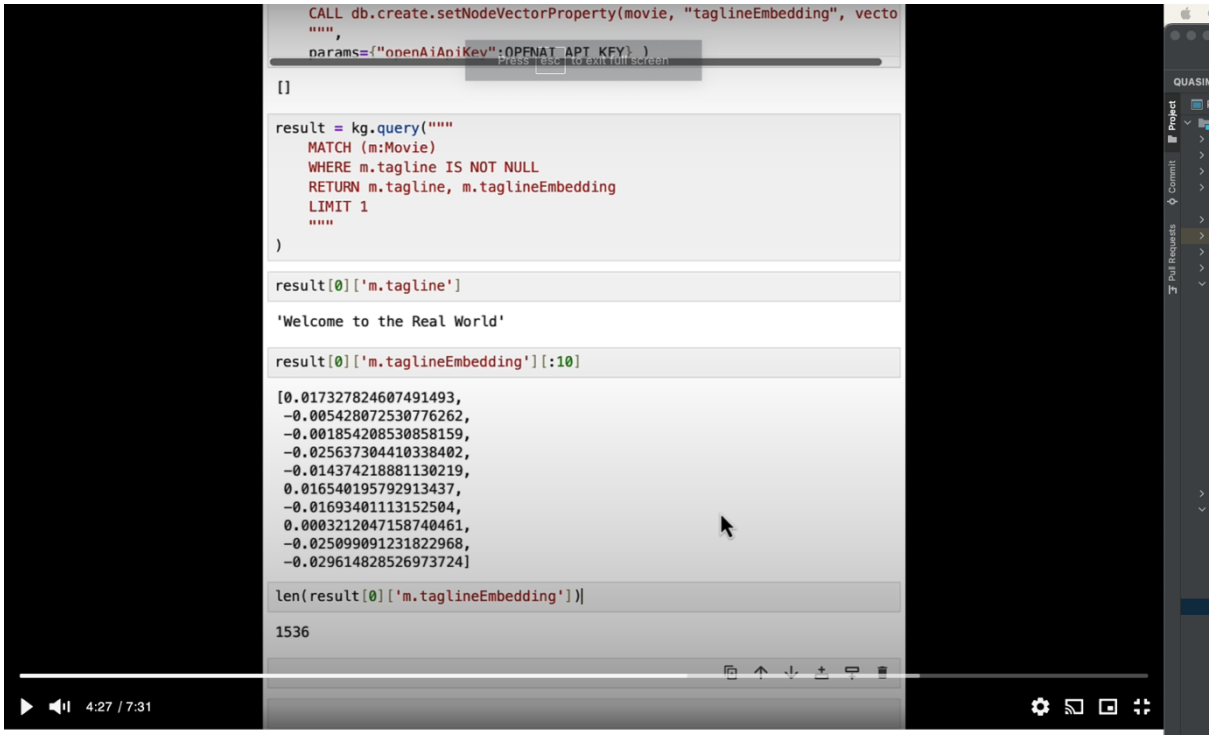
Creating a vector index if not already exist.
Index on tagline
Adding some configs.



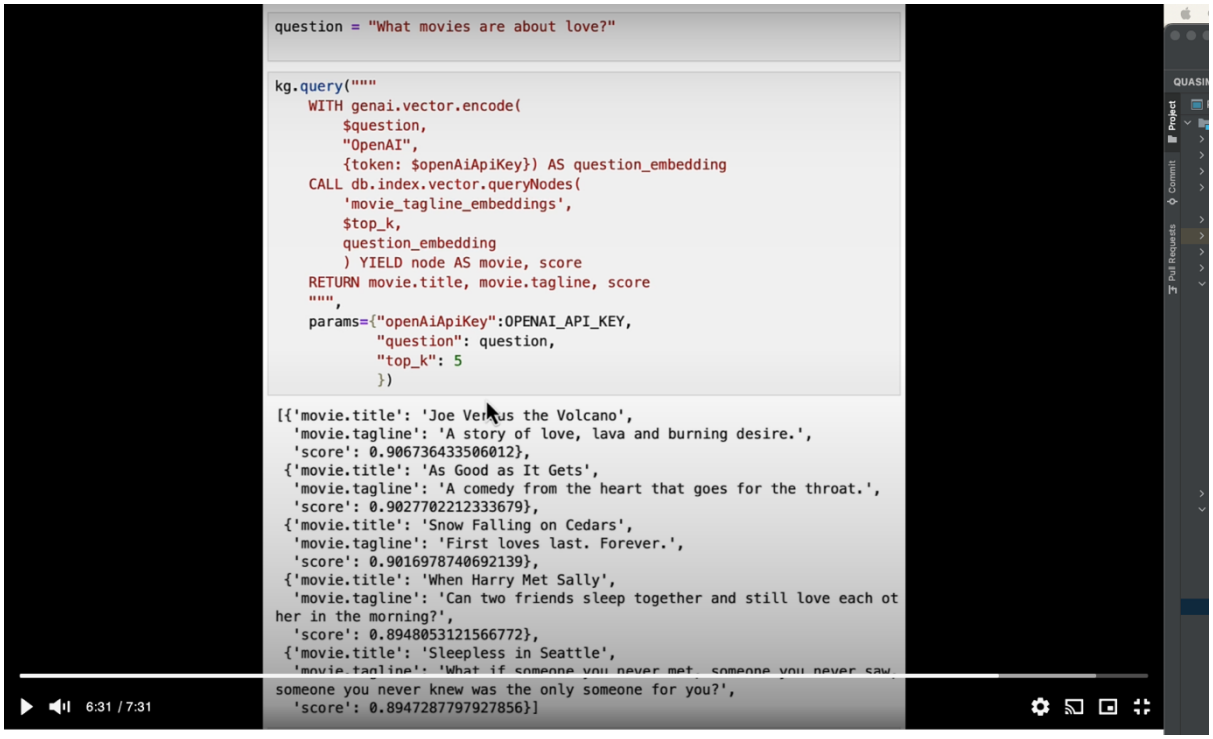
Running the openai encoder to get embeddings and store them as a vector index.
In the cypher query, we can pass a value as params.
We directly rum the code to generate the encodings within the query, to do it for all taglines for alll movies.



Looking at the stored embeddings:


CALL db.create.setNodeVectorProperty(movie, "taglineEmbedding", vecto
"""
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
4:27 / 7:31

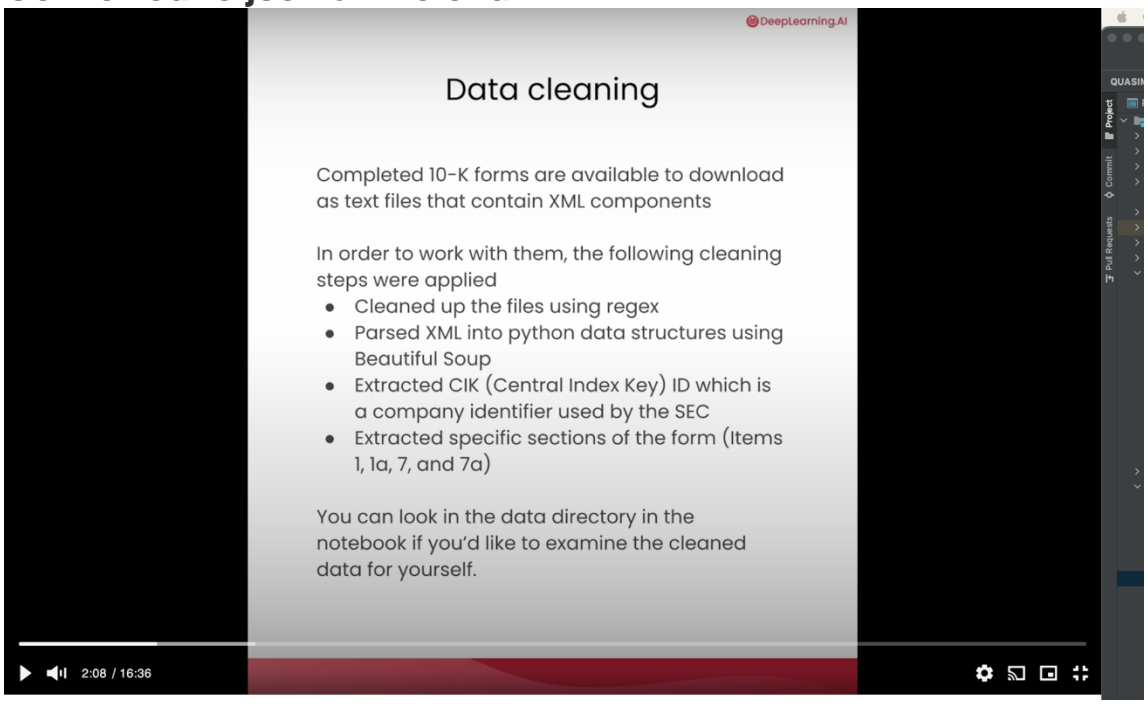
Query:

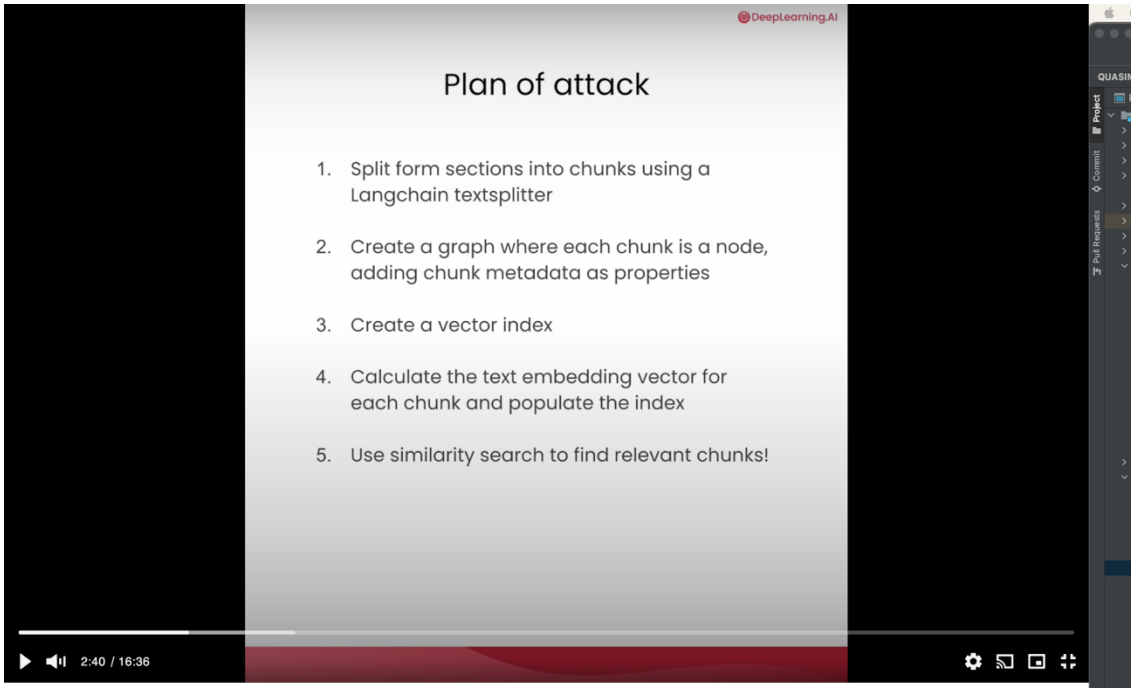

question = "What movies are about love?"
kg.query("""
WITH genai.vector.encode(
$question,
"OpenAI",
{token: $openAiApiKey}) AS question_embedding
CALL db.index.vector.queryNodes(
'movie_tagline_embeddings',
$top_k,
question_embedding
) YIELD node AS movie, score
RETURN movie.title, movie.tagline, score
""",
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 ot
her in the morning?',
'score': 0.8948053121566772},
{'movie.title': 'Sleepless in Seattle',
someone you never knew was the only someone for you?',
'score': 0.8947287797927856}]
6:31 / 7:31

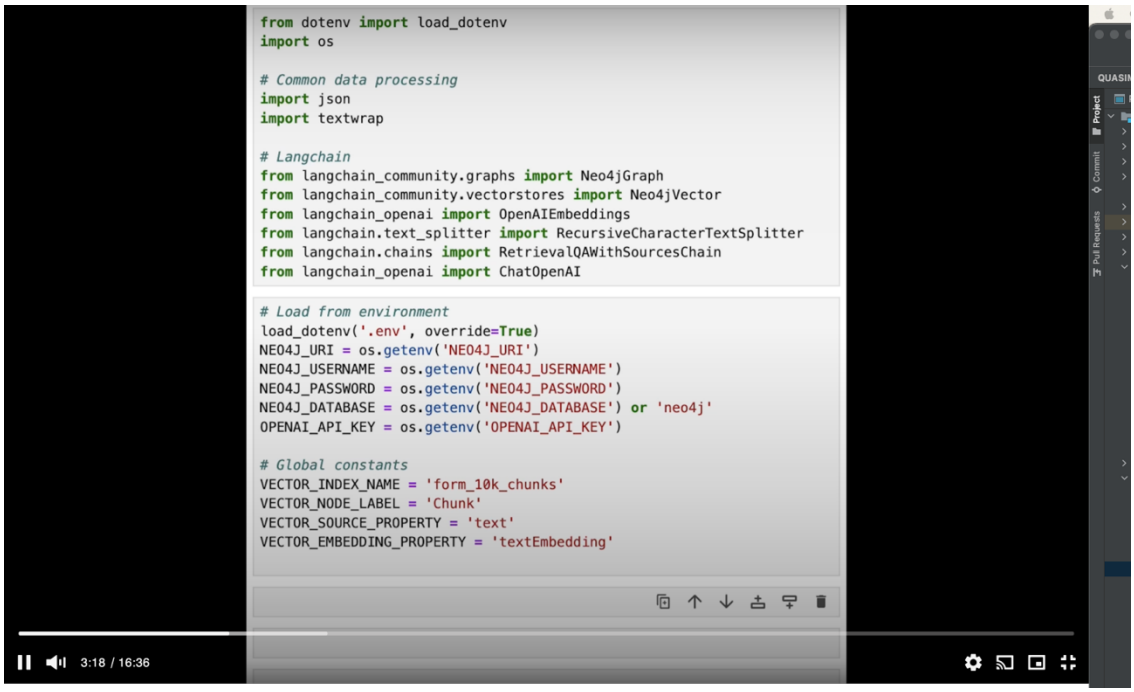
Lesson 4: Constructing a Knowledge Graph from Text Documents

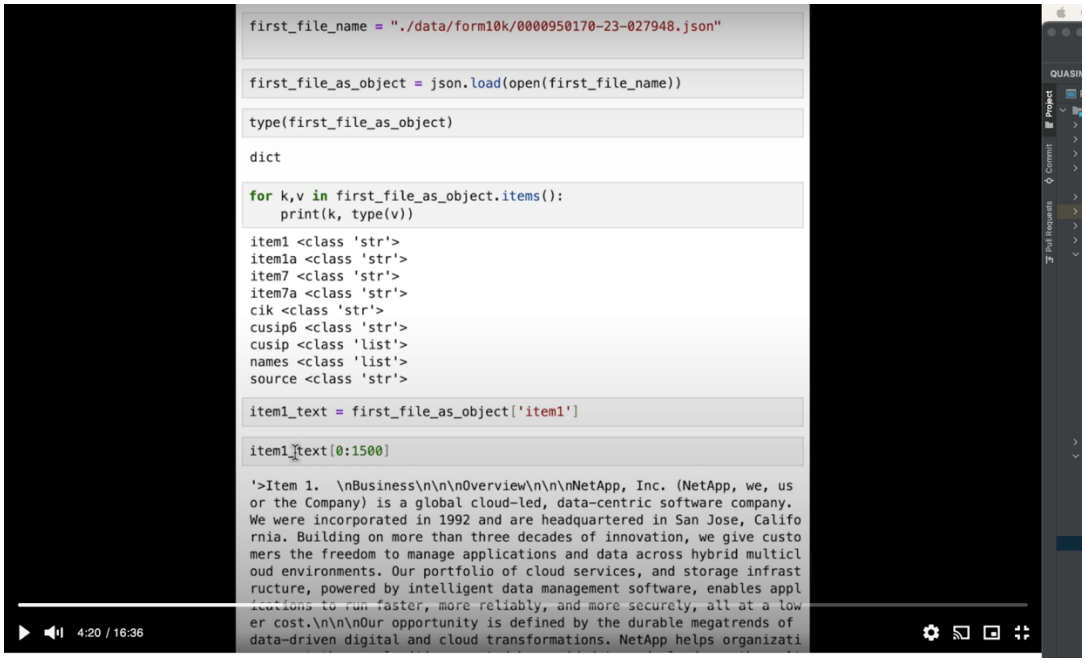
Cleaning:

Converted to json at the end.

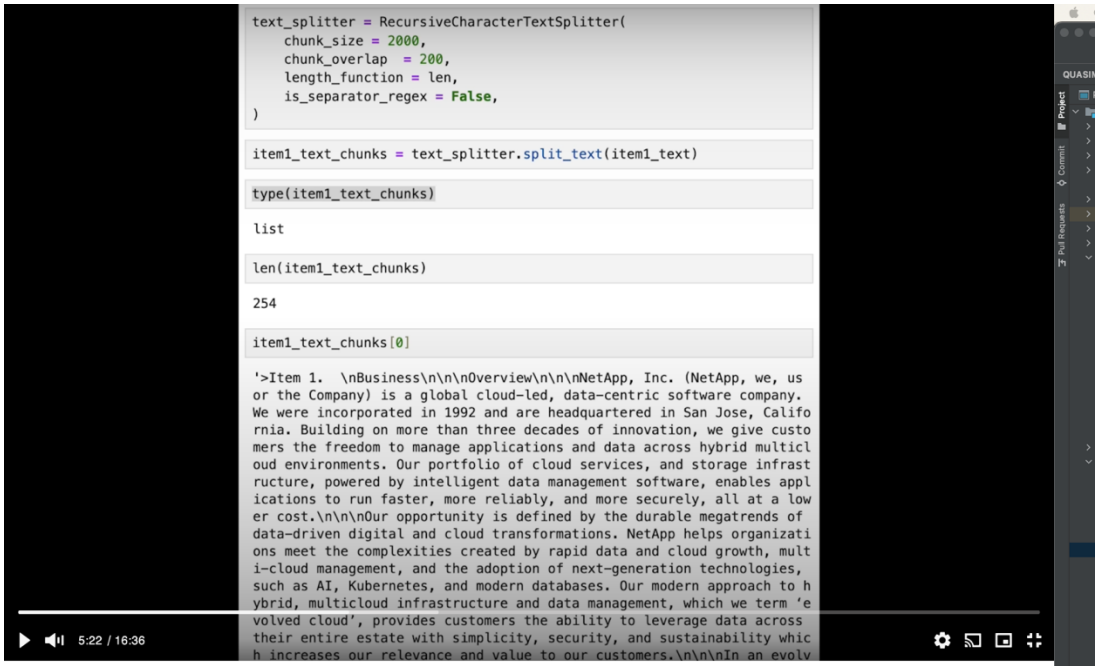



DeepLearning.AI
Plan of attack
1. Split form sections into chunks using a Langchain textsplitter
2. Create a graph where each chunk is a node, adding chunk metadata as properties
3. Create a vector index
4. Calculate the text embedding vector for each chunk and populate the index
5. Use similarity search to find relevant chunks!
2:40 / 16:36

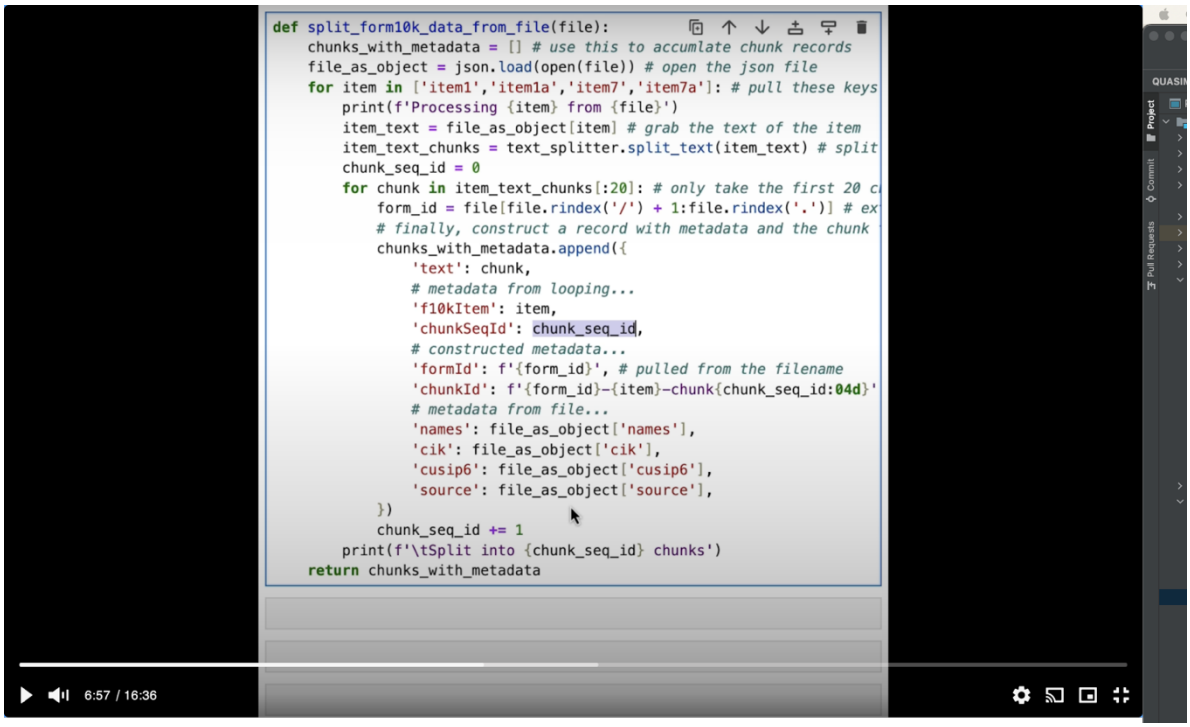

from dotenv import load_dotenv
import os
# Common data processing
import json
import textwrap
# Langchain
from langchain_community.graphs import Neo4jGraph
from langchain_community.vectorstores import Neo4jVector
from langchain_openai import OpenAIEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
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'
3:18 / 16:36

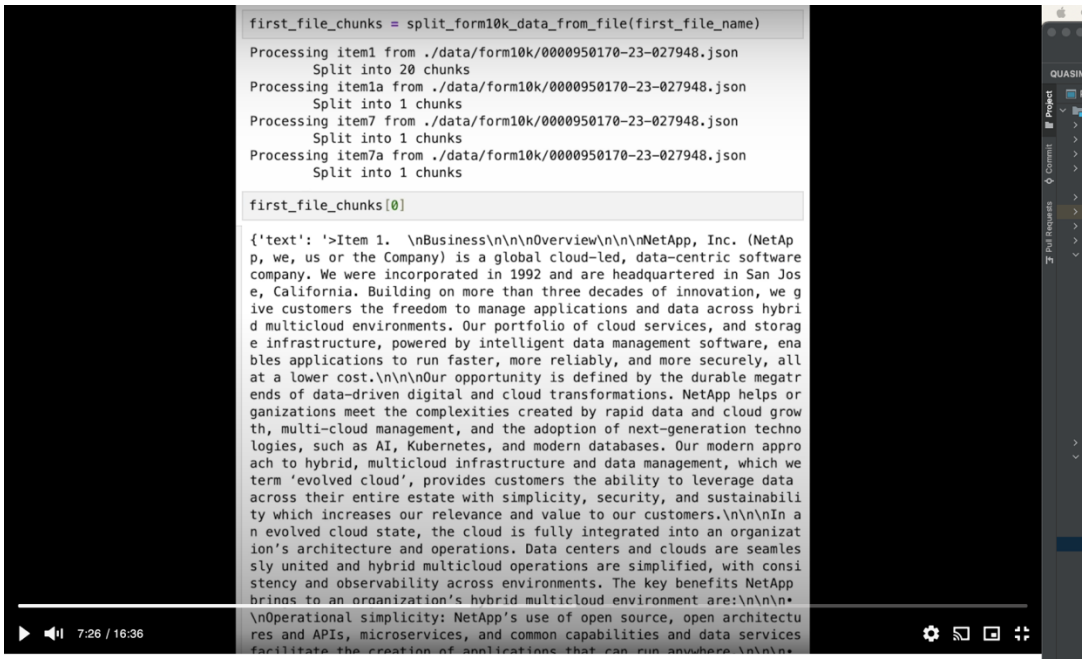


Creating chunks, using langchain text splitter.



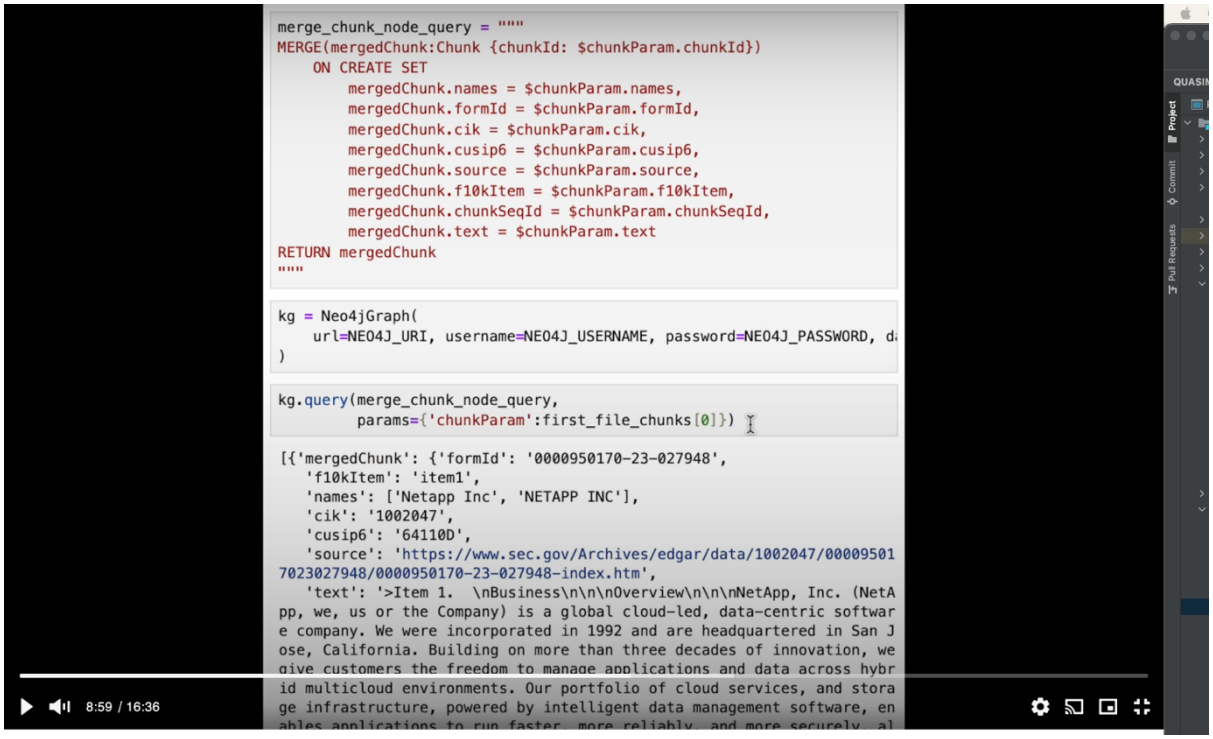
Helper function to parse the data, create chunks and return chunks + metadata.



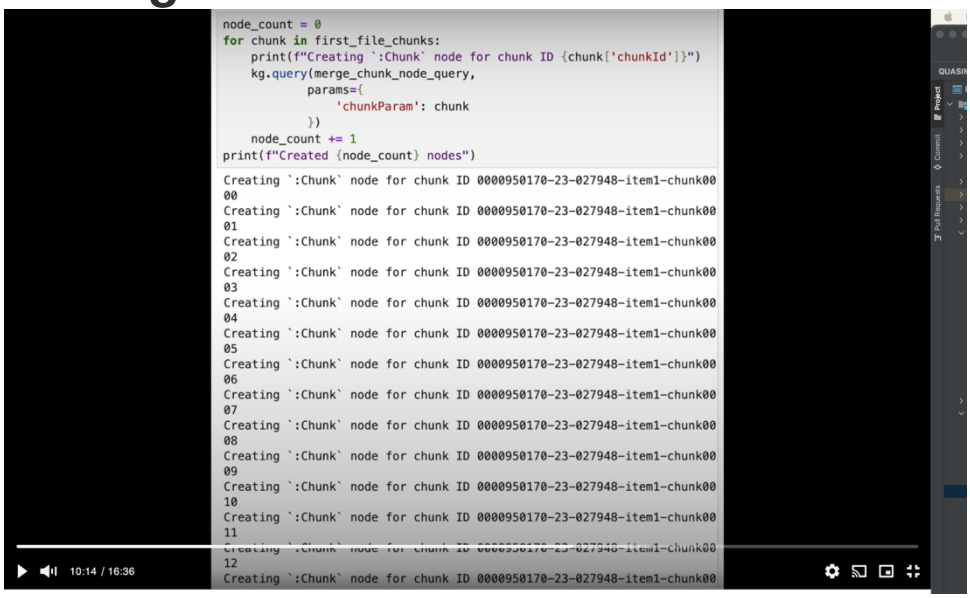


Adding chunk to KG as a node.

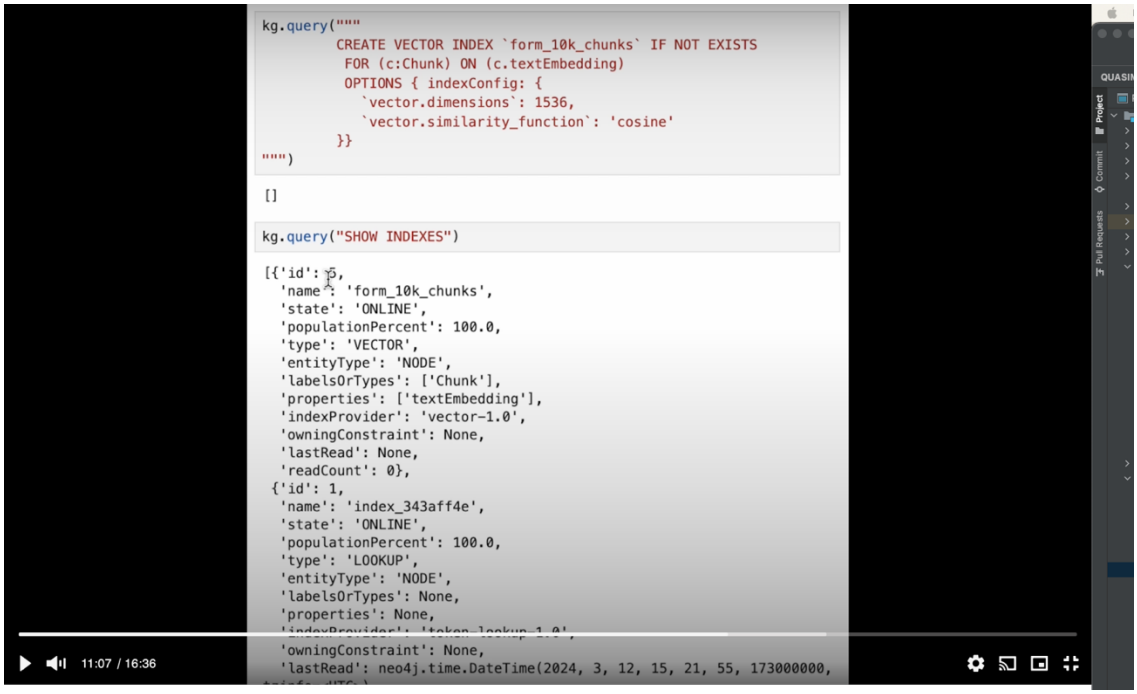
Creating another index: which makes a unique constraint on chunk id so that it doesnt repeat.



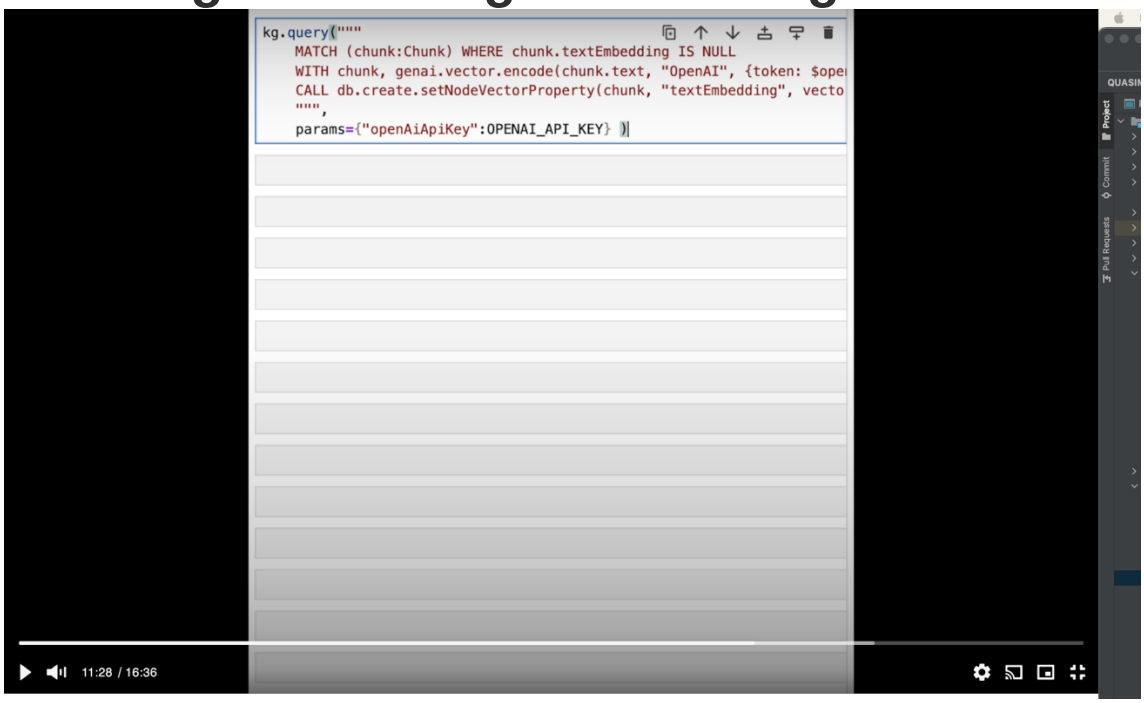
Adding all chunks as node.

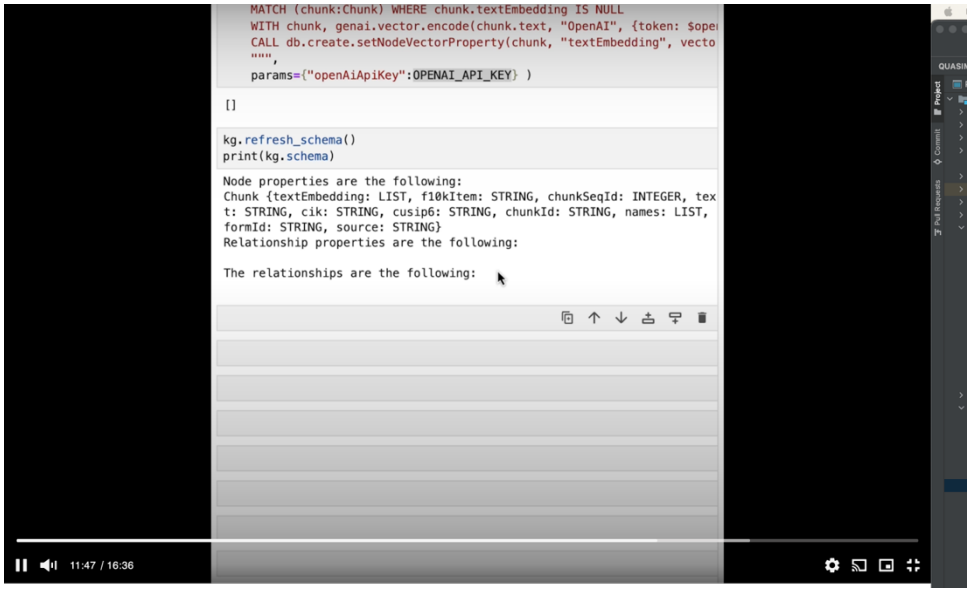


Creating vector index to store embeddings of text in these chunks:

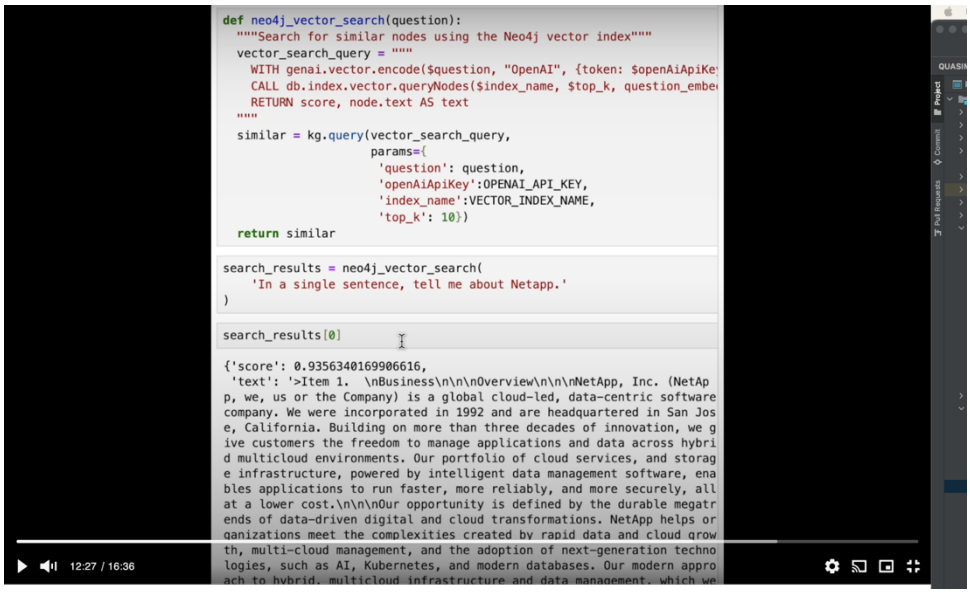


creating embeddings and storing in nodes:

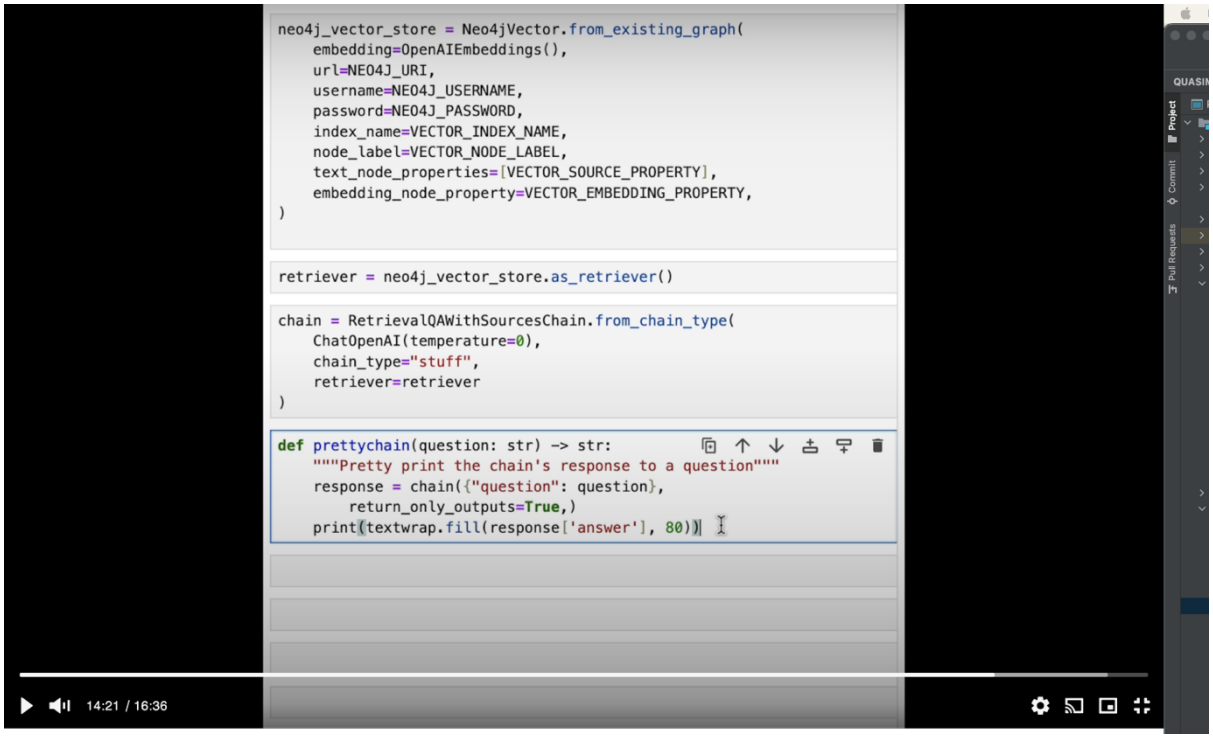




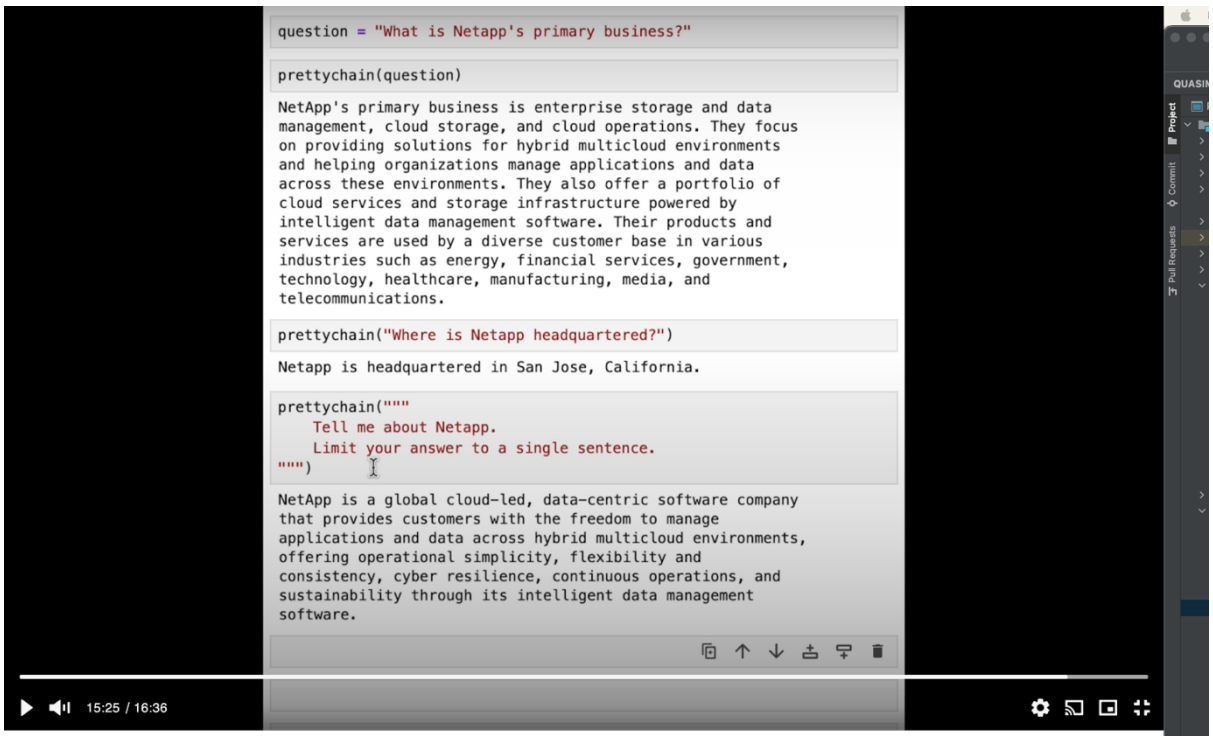
Question embedding and searching:



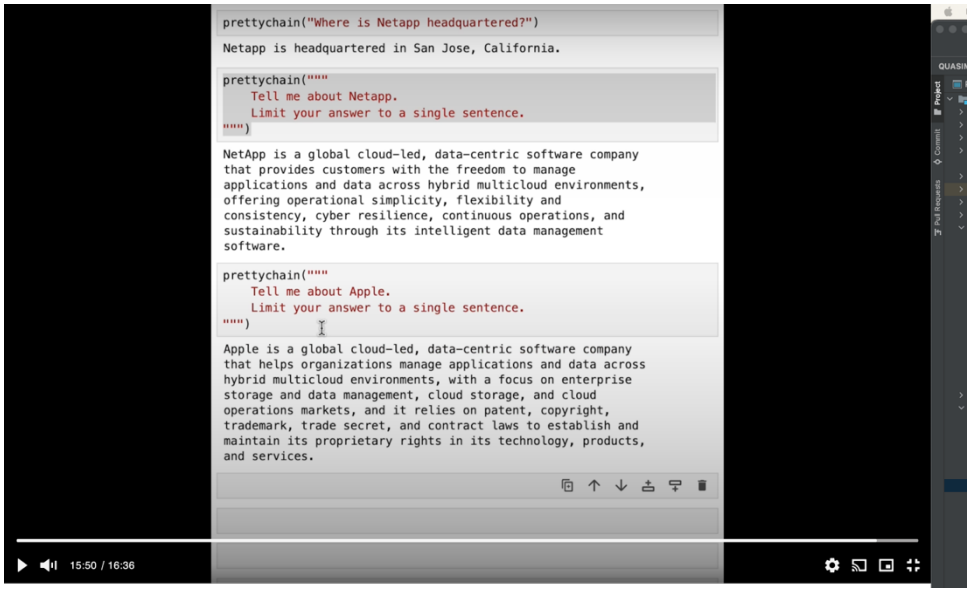
Now, we use neo4j as a vector DB, with the help of Langchain, we convert it as a retriever. Using OpenAI for chat.
And another chain for OpenAI chat again provided by Langchain:


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))
14:21 / 16:36

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.
15:25 / 16:36

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



Simple prompt engineering:

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