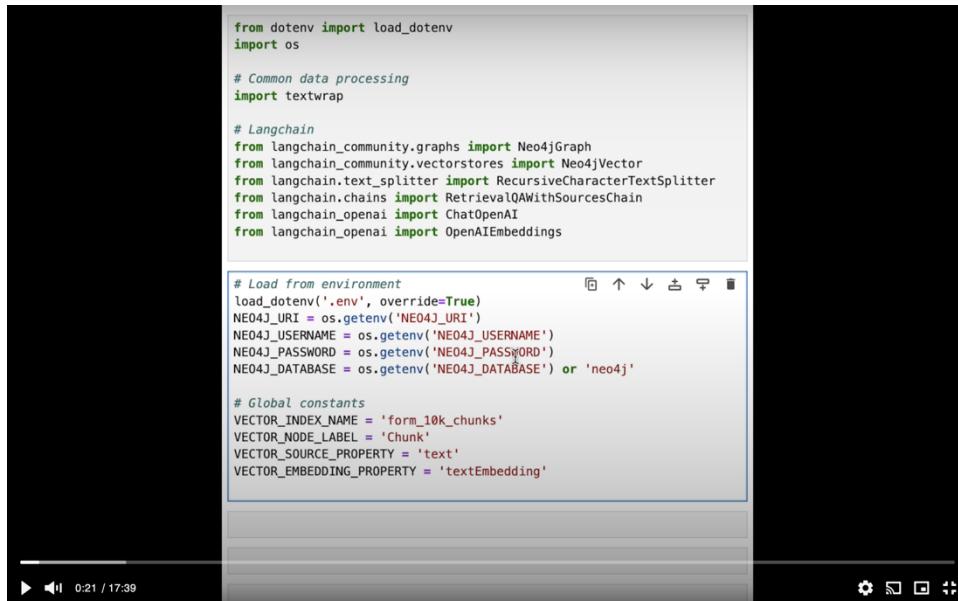


Lesson 5: Adding Relationships to the SEC Knowledge Graph



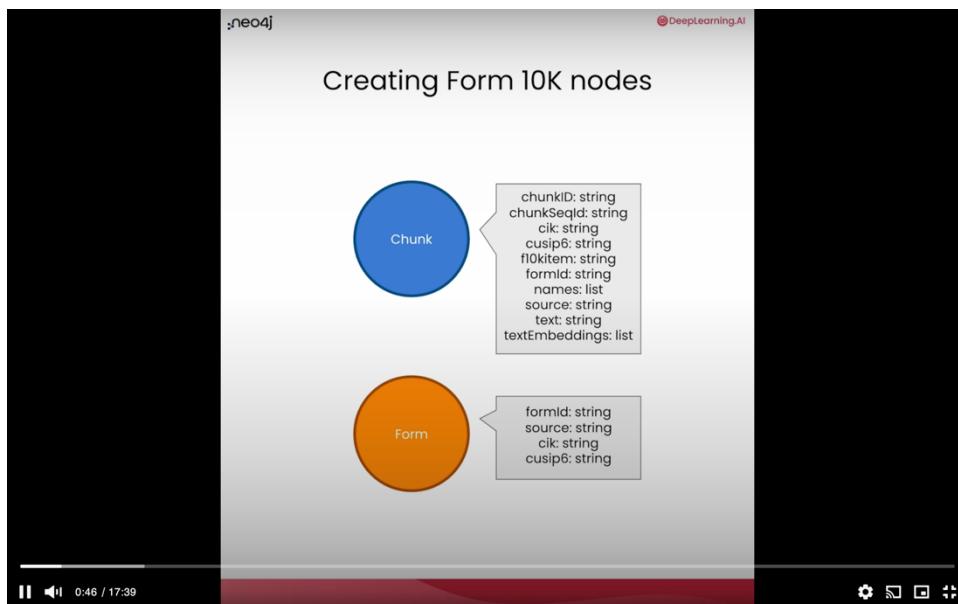
```
from dotenv import load_dotenv
import os

# Common data processing
import textwrap

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

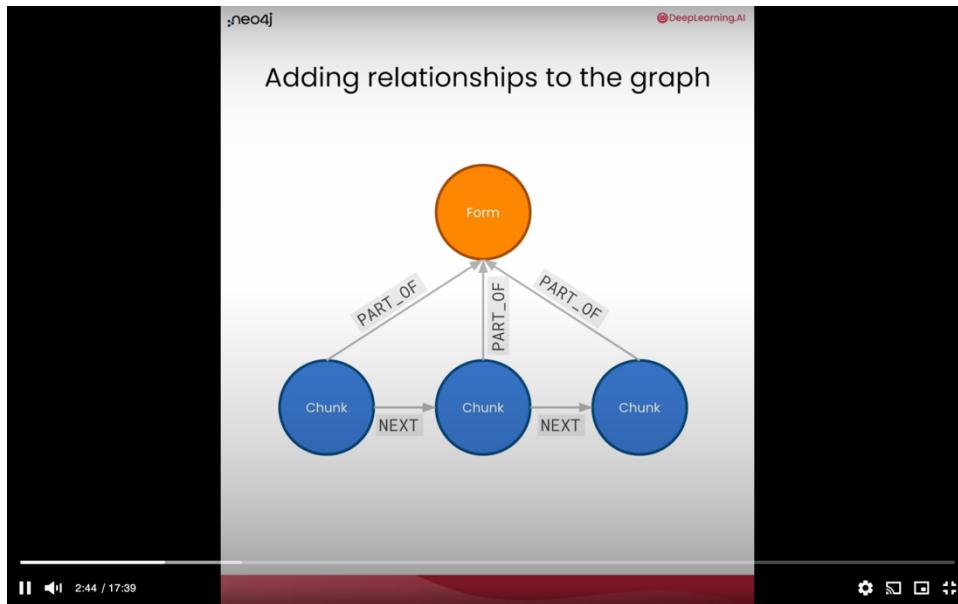
# Load from environment
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE') or 'neo4j'

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



```
cypher = """  
MATCH (anyChunk:Chunk)  
WITH anyChunk LIMIT 1  
RETURN anyChunk { .names, .source, .formId, .cik, .cusip6 } as formInfo  
""";  
form_info_list = kg.query(cypher)  
  
form_info_list  
  
[{'formInfo': {'cik': '1002047',  
'source': 'https://www.sec.gov/Archives/edgar/data/1002047/0000950170/  
23027948/0000950170-23-027948-index.htm',  
'formId': '0000950170-23-027948',  
'names': ['Netapp Inc', 'NETAPP INC'],  
'cusip6': '64110D'}}]  
form_info = form_info_list[0]['formInfo']  
  
form_info  
  
{'cik': '1002047',  
'source': 'https://www.sec.gov/Archives/edgar/data/1002047/0000950170/  
23027948/0000950170-23-027948-index.htm',  
'formId': '0000950170-23-027948',  
'names': ['Netapp Inc', 'NETAPP INC'],  
'cusip6': '64110D'}  
  
▶ 🔍 1:41 / 17:39
```

```
cypher = """  
MERGE (f:Form {formId: $formInfoParam.formId })  
ON CREATE  
    SET f.names = $formInfoParam.names  
    SET f.source = $formInfoParam.source  
    SET f.cik = $formInfoParam.cik  
    SET f.cusip6 = $formInfoParam.cusip6  
""";  
  
kg.query(cypher, params={'formInfoParam': form_info})  
[]  
  
kg.query("MATCH (f:Form) RETURN count(f) as formCount")  
[{'formCount': 1}]  
  
▶ 🔍 2:24 / 17:39
```



So chunks are in linkedlist.

Now lets query all chunks from a particular form id:

now we want chunkseqid in ascending order, as linkedlist, so order by.

And also from same section.

So. ascending chunks id, within each section,

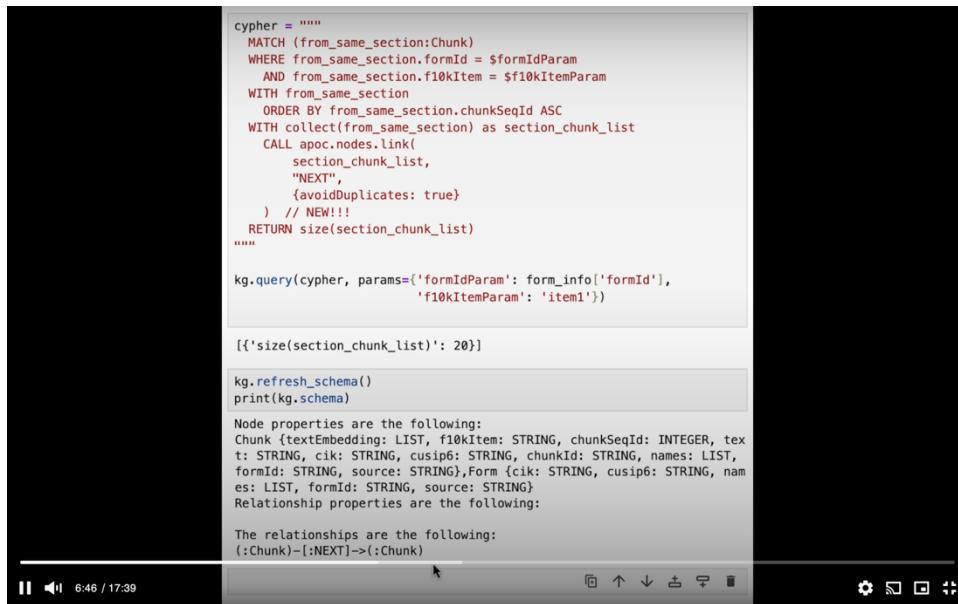
```
cypher = """  
MATCH (from_same_section:Chunk)  
WHERE from_same_section.formId = $formIdParam  
    AND from_same_section.f10kItem = $f10kItemParam // NEW!!!  
RETURN from_same_section { .formId, .f10kItem, .chunkId, .chunkSeqId  
    ORDER BY from_same_section.chunkSeqId ASC  
    LIMIT 10  
....  
  
kg.query(cypher, params={'formIdParam': form_info['formId'],  
                        'f10kItemParam': 'item1'})  
  
[{'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0000',  
                      'chunkSeqId': 0},  
 'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0001',  
                      'chunkSeqId': 1},  
 'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0002',  
                      'chunkSeqId': 2},  
 'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0003',  
                      'chunkSeqId': 3},  
 'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0004',  
                      'chunkSeqId': 4},  
 'from_same_section': {'formId': '0000950170-23-027948',  
                      'f10kItem': 'item1',  
                      'chunkId': '0000950170-23-027948-item1-chunk0005',  
                      'chunkSeqId': 5}],  
 4:56 / 17:39
```

We use COLLECT to get the output as a list. With one row.

```
cypher = """  
MATCH (from_same_section:Chunk)  
WHERE from_same_section.formId = $formIdParam  
    AND from_same_section.f10kItem = $f10kItemParam  
WITH from_same_section { .formId, .f10kItem, .chunkId, .chunkSeqId }  
    ORDER BY from_same_section.chunkSeqId ASC  
    LIMIT 10  
RETURN collect(from_same_section) // NEW!!!  
....  
  
kg.query(cypher, params={'formIdParam': form_info['formId'],  
                        'f10kItemParam': 'item1'})  
  
[{'collect(from_same_section)': [{"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0000',  
                                 'chunkSeqId': 0},  
                               {"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0001',  
                                 'chunkSeqId': 1},  
                               {"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0002',  
                                 'chunkSeqId': 2},  
                               {"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0003',  
                                 'chunkSeqId': 3},  
                               {"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0004',  
                                 'chunkSeqId': 4},  
                               {"formId": "0000950170-23-027948",  
                                 'f10kItem': 'item1',  
                                 'chunkId': '0000950170-23-027948-item1-chunk0005',  
                                 'chunkSeqId': 5}],  
 5:44 / 17:39
```

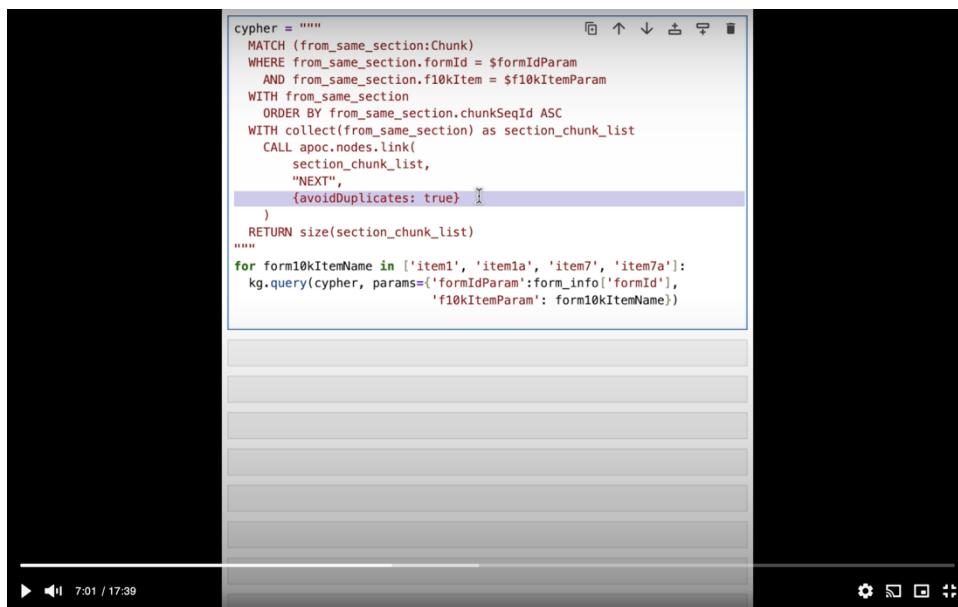
Now we add relationship for these chunks:

```
cypher = """  
MATCH (from_same_section:Chunk)  
WHERE from_same_section.formId = $formIdParam  
    AND from_same_section.f10kItem = $f10kItemParam  
WITH from_same_section  
ORDER BY from_same_section.chunkSeqId ASC  
WITH collect(from_same_section) as section_chunk_list  
CALL apoc.nodes.link(  
    section_chunk_list,  
    "NEXT",  
    {avoidDuplicates: true}  
) // NEW!!!  
RETURN size(section_chunk_list)  
""";  
  
kg.query(cypher, params={'formIdParam': form_info['formId'],  
'f10kItemParam': 'item1'})  
  
[{'size(section_chunk_list)': 20}]  
  
kg.refresh_schema()  
print(kg.schema)  
  
Node properties are the following:  
Chunk {textEmbedding: LIST, f10kItem: STRING, chunkSeqId: INTEGER, tex  
t: STRING, cik: STRING, cusip6: STRING, chunkId: STRING, names: LIST,  
formId: STRING, source: STRING}, Form {cik: STRING, cusip6: STRING, nam  
es: LIST, formId: STRING, source: STRING}  
Relationship properties are the following:  
The relationships are the following:  
(:Chunk)-[:NEXT]-(:Chunk)
```



Now we do that für all items:

```
cypher = """  
MATCH (from_same_section:Chunk)  
WHERE from_same_section.formId = $formIdParam  
    AND from_same_section.f10kItem = $f10kItemParam  
WITH from_same_section  
ORDER BY from_same_section.chunkSeqId ASC  
WITH collect(from_same_section) as section_chunk_list  
CALL apoc.nodes.link(  
    section_chunk_list,  
    "NEXT",  
    {avoidDuplicates: true})  
) // NEW!!!  
RETURN size(section_chunk_list)  
""";  
  
for form10kItemName in ['item1', 'item1a', 'item7', 'item7a']:  
    kg.query(cypher, params={'formIdParam':form_info['formId'],  
    'f10kItemParam': form10kItemName})
```



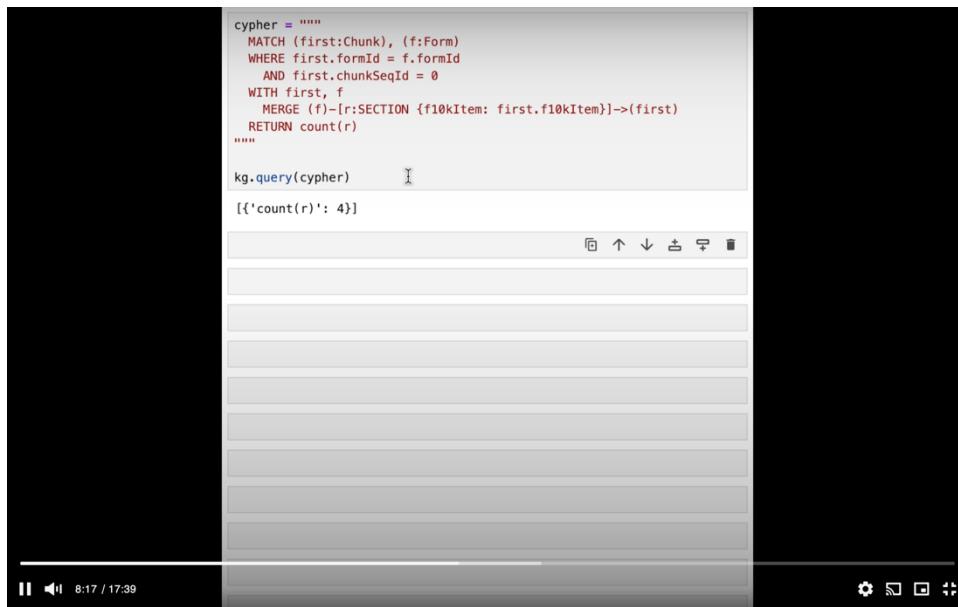
Relationship between chunks and form it belongs to:

```
cypher = """  
MATCH (c:Chunk), (f:Form)  
WHERE c.formId = f.formId  
MERGE (c)-[newRelationship:PART_OF]-(f)  
RETURN count(newRelationship)  
"""  
  
kg.query(cypher)  
[{'count(newRelationship)': 23}]
```



Linking first chunk to forms. To identify diff sections_

```
cypher = """  
MATCH (first:Chunk), (f:Form)  
WHERE first.formId = f.formId  
AND first.chunkSeqId = 0  
WITH first, f  
MERGE (f)-[r:SECTION {f10kItem: first.f10kItem}]->(first)  
RETURN count(r)  
"""  
  
kg.query(cypher)  
[{'count(r)': 4}]
```



Query-_-:

Example cypher queries

```
cypher = """  
MATCH (f:Form)-[r:SECTION]-(first:Chunk)  
WHERE f.formId = $formIdParam  
    AND r.f10kItem = $f10kItemParam  
RETURN first.chunkId as chunkId, first.text as text  
"""  
  
first_chunk_info = kg.query(cypher, params={  
    'formIdParam': form_info['formId'],  
    'f10kItemParam': 'item1'  
})[0]  
  
first_chunk_info
```

{'chunkId': '0000950170-23-027948-item1-chunk0000',
'text': 'Item 1. \nBusiness\\n\\nOverview\\n\\nNetApp, Inc. (NetAp
p, we, us or the Company) is a global cloud-led, data-centric software
company. We were incorporated in 1992 and are headquartered in San Jos
e, California. Building on more than three decades of innovation, we g
ive customers the freedom to manage applications and data across hybr
d multicloud environments. Our portfolio of cloud services, and storag
e infrastructure, powered by intelligent data management software, ena
bles applications to run faster, more reliably, and more securely, all
at a lower cost.\\n\\nOur opportunity is defined by the durable megatr
ends of data-driven digital and cloud transformations. NetApp helps or
ganizations meet the complexities created by rapid data and cloud grow
th, multi-cloud management, and the adoption of next-generation technolo
gies, such as AI, Kubernetes, and modern databases. Our modern appro
ach 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 sustainabilit
y, which increases our relevance and value to our customers.\\n\\nIn a
few years, we expect to have 100 billion objects in our storage system, and
we will be managing petabytes of data in the cloud. This is a significant
shift from our traditional on-premises storage solutions, and it requires
a new way of thinking about data management. We believe that our
innovative solutions will help organizations succeed in this new era of
data management.\\n\\nWe are committed to providing the best possible
experience for our customers, and we are constantly working to improve
our products and services. We are also focused on sustainability, and we
are committed to reducing our environmental impact. We believe that
by doing so, we can help protect the planet for future generations.
\\n\\nThank you for your interest in NetApp. We hope you find this
information useful, and we look forward to working with you.
\\n\\nBest regards,
\\n\\n[Your Name]
\\n\\n[Your Title]
\\n\\n[Your Company]
\\n\\n[Your Email]
\\n\\n[Your Phone Number]

With chunk_id: find next chunk t it:

```
cypher = """  
    MATCH (first:Chunk)-[:NEXT]->(nextChunk:Chunk)  
        WHERE first.chunkId = $chunkIdParam  
    RETURN nextChunk.chunkId as chunkId, nextChunk.text as text  
"""  
  
next_chunk_info = kg.query(cypher, params={  
    'chunkIdParam': first_chunk_info['chunkId']  
})[0]  
  
next_chunk_info  
  
{'chunkId': '0000950170-23-027948-item1-chunk0001',  
 'text': "\nflexibility and consistency: NetApp makes moving data and  
 applications between environments seamless through a common storage fo  
 undation across on-premises and multicloud environments.\n\n\n\n\nCyber  
 resilience: NetApp unifies monitoring, data protection, security, gove  
 rnance, and compliance for total cyber resilience – with consistency a  
 nd automation across environments. \n\n\n\n\nContinuous operations: Net  
 App uses AI-driven automation for continuous optimization to service a  
 pplications and store stateless and stateful applications at the lowes  
 t possible costs.\n\n\n\n\nSustainability: NetApp has industry-leading  
 tools to audit consumption, locate waste, and set guardrails to stop o  
 verprovisioning.\n\n\n\nProduct, Solutions and Services Portfolio\n\n\n\n\nNetApp's portfolio of cloud services and storage infrastructure is p  
 owered by intelligent data management software. Our operations are org  
 anized into two segments: Hybrid Cloud and Public Cloud.\n\n\n\n\nHy  
 brid Cloud\n\n\n\n\nHybrid Cloud\noffers a portfolio of storage manageme  
 nt and infrastructure solutions that help customers recast their tradi  
 tional data centers into modern data centers with the power of the clo  
 ud. Our hybrid cloud portfolio is designed to operate with public clou  
 ds to unlock the potential of hybrid, multi-cloud operations. We offer  
 a broad portfolio of cloud-connected all-flash, hybrid-flash, and obje  
 ct storage systems, powered by intelligent data management software. H  
 ybrid Cloud is composed of software, hardware, and related support, as  
 well as professional and other services.\n\n\n\nIntelligent data managem
```

Get one before and one later chunks, and also path, or window::

A screenshot of a Cypher editor interface. The code area contains the following Cypher script:

```
print(first_chunk_info['chunkId'], next_chunk_info['chunkId'])
0000950170-23-027948-item1-chunk0000 0000950170-23-027948-item1-chunk001
cypher = """
    MATCH (c1:Chunk)-[:NEXT]->(c2:Chunk)-[:NEXT]->(c3:Chunk)
    WHERE c2.chunkId = $chunkIdParam
    RETURN c1.chunkId, c2.chunkId, c3.chunkId
"""

kg.query(cypher,
    params={'chunkIdParam': next_chunk_info['chunkId']})

[{'c1.chunkId': '0000950170-23-027948-item1-chunk0000',
  'c2.chunkId': '0000950170-23-027948-item1-chunk0001',
  'c3.chunkId': '0000950170-23-027948-item1-chunk0002'}]

cypher = """
    MATCH window = (c1:Chunk)-[:NEXT]->(c2:Chunk)-[:NEXT]->(c3:Chunk)
    WHERE c2.chunkId = $chunkIdParam
    RETURN length(window) as windowPathLength
"""

kg.query(cypher,
    params={'chunkIdParam': next_chunk_info['chunkId']})
[{'windowPathLength': 2}]
```

The status bar at the bottom shows "10:47 / 17:39". The interface includes standard file, edit, and search tools.

V

if we try to match first chunk, and ask for one before chunk, that it wont return anything:

A screenshot of a Cypher editor interface. The code area contains the following Cypher script:

```
cypher = """
    MATCH window = (c1:Chunk)-[:NEXT]->(c2:Chunk)-[:NEXT]->(c3:Chunk)
    WHERE c2.chunkId = $chunkIdParam
    RETURN length(window) as windowPathLength
"""

kg.query(cypher,
    params={'chunkIdParam': first_chunk_info['chunkId']})

[]
```

The status bar at the bottom shows "11:23 / 17:39". The interface includes standard file, edit, and search tools.

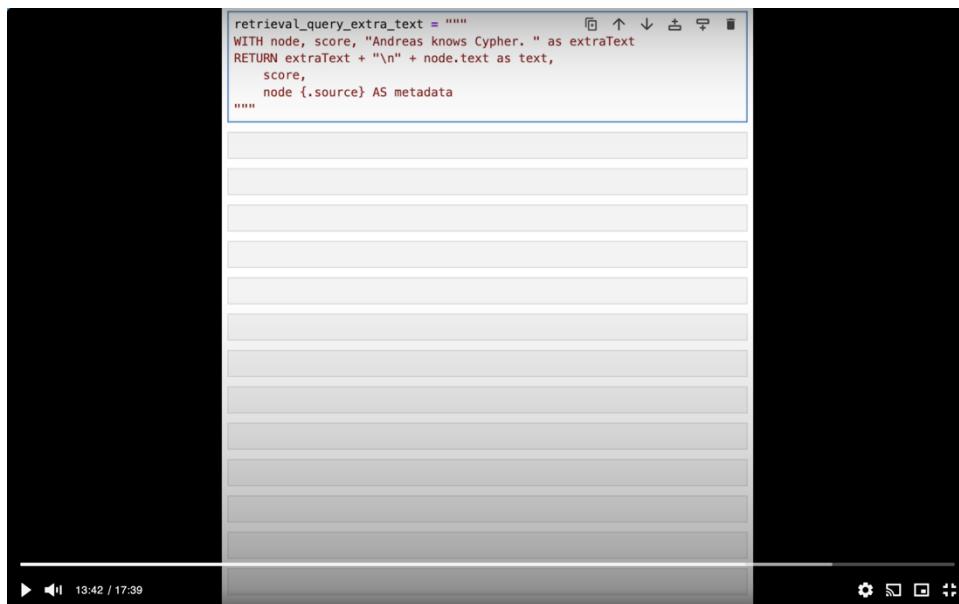
How to solve it, range

Min 0, max 1 match

```
cypher = """  
MATCH windows=  
  (:Chunk)-[:NEXT*0..1]->(c:Chunk)-[:NEXT*0..1]->(:Chunk)  
  WHERE c.chunkId = $chunkIdParam  
  RETURNN length(window)  
  """  
  
kg.query(cypher,  
         params={'chunkIdParam': first_chunk_info['chunkId']})  
  
[{'length(window)': 0}, {'length(window)': 1}]
```

```
cypher = """  
MATCH windows=  
    (:Chunk)-[:NEXT*0..1]-(c:Chunk)-[:NEXT*0..1]-(:Chunk)  
    WHERE c.chunkId = $chunkIdParam  
    WITH window as longestChunkWindow  
    ORDER BY length(window) DESC LIMIT 1  
RETURN length(longestChunkWindow)  
"""  
  
kg.query(cypher,  
    params={'chunkIdParam': first_chunk_info['chunkId']})  
  
[{'length(longestChunkWindow)': 1}]
```

```
retrieval_query_extra_text = """  
WITH node, score, "Andreas knows Cypher. " as extraText  
RETURN extraText + "\n" + node.text as text,  
      score,  
      node {.source} AS metadata  
"""
```



A screenshot of a code editor window. The main area contains a single line of Cypher code. The status bar at the bottom shows the time as 13:42 / 17:39.

```
retrieval_query_extra_text = """  
WITH node, score, "Andreas knows Cypher. " as extraText  
RETURN extraText + "\n" + node.text as text,  
      score,  
      node {.source} AS metadata  
"""  
  
vector_store_extra_text = Neo4jVector.from_existing_index(  
    embedding=OpenAIEMBEDDINGS(),  
    url=NEO4J_URI,  
    username=NEO4J_USERNAME,  
    password=NEO4J_PASSWORD,  
    database="neo4j",  
    index_name=VECTOR_INDEX_NAME,  
    text_node_property=VECTOR_SOURCE_PROPERTY,  
    retrieval_query=retrieval_query_extra_text, # NEW !!!  
)  
  
# Create a retriever from the vector store  
retriever_extra_text = vector_store_extra_text.as_retriever()  
  
# Create a chatbot Question & Answer chain from the retriever  
chain_extra_text = RetrievalQAWithSourcesChain.from_chain_type(  
    ChatOpenAI(temperature=0),  
    chain_type="stuff",  
    retriever=retriever_extra_text
```



A screenshot of a code editor window. The main area contains a multi-line Python script. The code defines a variable `retrieval_query_extra_text` with a Cypher query, and another variable `vector_store_extra_text` which is an instance of `Neo4jVector` with specific parameters. It also imports `OpenAIEMBEDDINGS`, `NEO4J_URI`, `NEO4J_USERNAME`, and `NEO4J_PASSWORD`. The script then creates a `retriever_extra_text` object using the vector store and finally creates a `chain_extra_text` object using `RetrievalQAWithSourcesChain` with `ChatOpenAI` and `stuff` chain type. The status bar at the bottom shows the time as 14:02 / 17:39.

A screenshot of a video player interface. The main content area is a code editor with the following Python code:

```
chain_extra_text(
    {"question": "What single topic does Andreas know about?",  
     return_only_outputs=True}  
  
{'answer': 'Andreas knows about Cypher.\n',  
 'sources': 'https://www.sec.gov/Archives/edgar/data/1002047/0000950170  
023027948/0000950170-23-027948-index.htm'}
```

The video player has a progress bar at the bottom left showing 15:00 / 17:39. On the right side, there are standard video control icons (play, pause, volume, etc.).

Now lets use window:

Lets create onne chain first without window to compare:

A screenshot of a video player interface. The main content area is a code editor with the following Python code:

```
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,  
)  
  
# Create a retriever from the vector store  
windowless_retriever = neo4j_vector_store.as_retriever()  
  
# Create a chatbot Question & Answer chain from the retriever  
windowless_chain = RetrievalQAWithSourcesChain.from_chain_type(  
    ChatOpenAI(temperature=0),  
    chain_type="stuff",  
    retriever=windowless_retriever  
)
```

The video player has a progress bar at the bottom left showing 15:25 / 17:39. On the right side, there are standard video control icons (play, pause, volume, etc.).

With window:

```
retrieval_query_window = """  
MATCH window=  
  (:Chunk)-[:NEXT*0..1]->(:node)-[:NEXT*0..1]->(:Chunk)  
WITH node, score, window as longestWindow  
  ORDER BY length(window) DESC LIMIT 1  
WITH nodes(longestWindow) as chunkList, node, score  
UNWIND chunkList as chunkRows  
WITH collect(chunkRows.text) as textList, node, score  
  RETURN apoc.text.join(textList, " \n ") as text,  
        score,  
        node {.source} AS metadata  
"""  
  
vector_store_window = Neo4jVector.from_existing_index(  
    embedding=OpenAIEmbeddings(),  
    url=NEO4J_URI,  
    username=NEO4J_USERNAME,  
    password=NEO4J_PASSWORD,  
    database="neo4j",  
    index_name=VECTOR_INDEX_NAME,  
    text_node_property=VECTOR_SOURCE_PROPERTY,  
    retrieval_query=retrieval_query_window, # NEW!!!  
)  
  
# Create a retriever from the vector store  
retriever_window = vector_store_window.as_retriever()  
  
# Create a chatbot Question & Answer chain from the retriever  
chain_window = RetrievalQAWithSourcesChain.from_chain_type(  
    ChatOpenAI(temperature=0),  
    chain_type="stuff",  
    retriever=retriever_window
```

Example:

```
question = "In a single sentence, tell me about Netapp's business."  
  
answer = windowless_chain(  
    {"question": question},  
    return_only_outputs=True,  
)  
print(textwrap.fill(answer["answer"]))  
  
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 in a hybrid multicloud world,  
and focusing primarily on the enterprise storage and data management,  
cloud storage, and cloud operations markets.  
  
answer = chain_window(  
    {"question": question},  
    return_only_outputs=True,  
)  
print(textwrap.fill(answer["answer"]))
```

Final part:

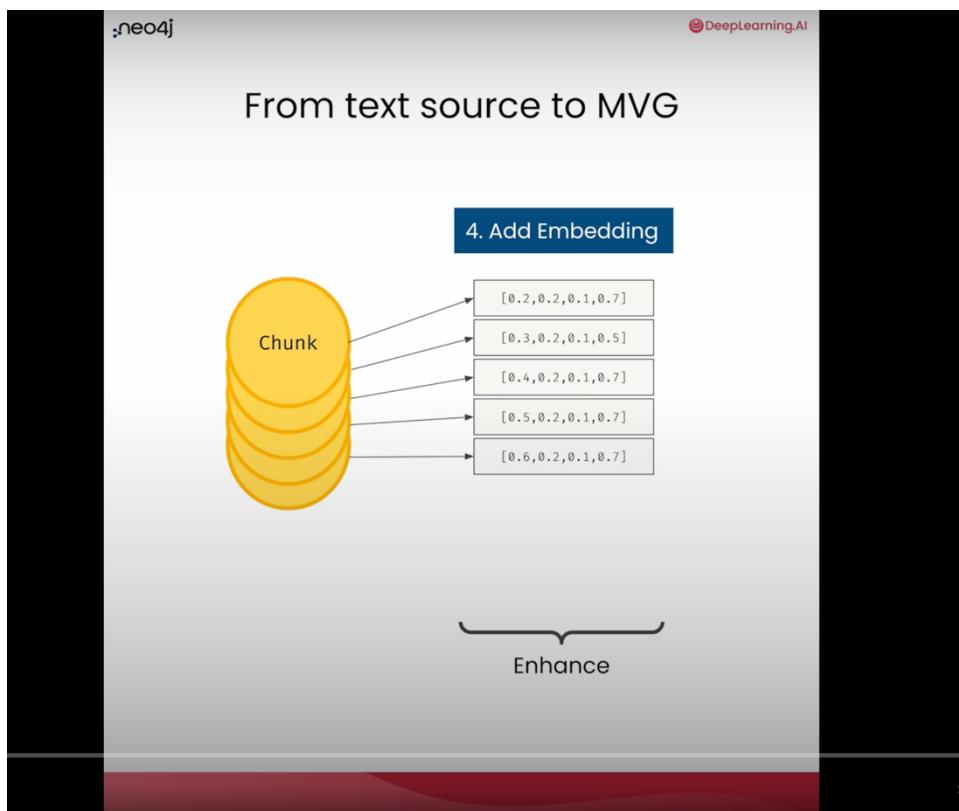
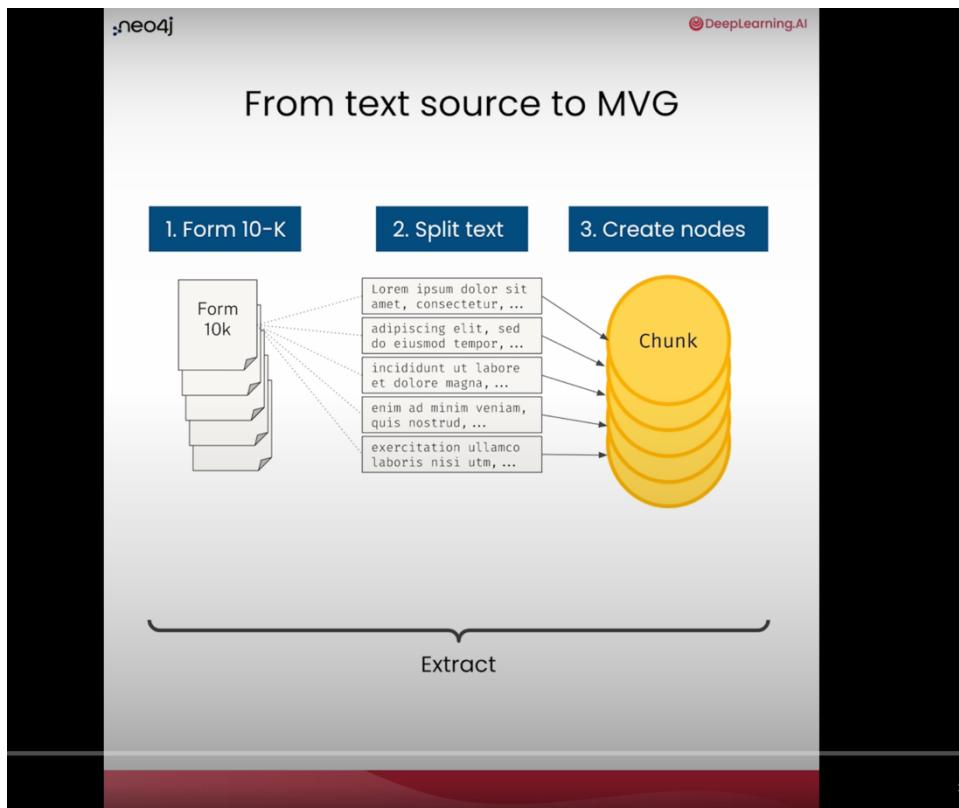
How did you create a knowledge graph?

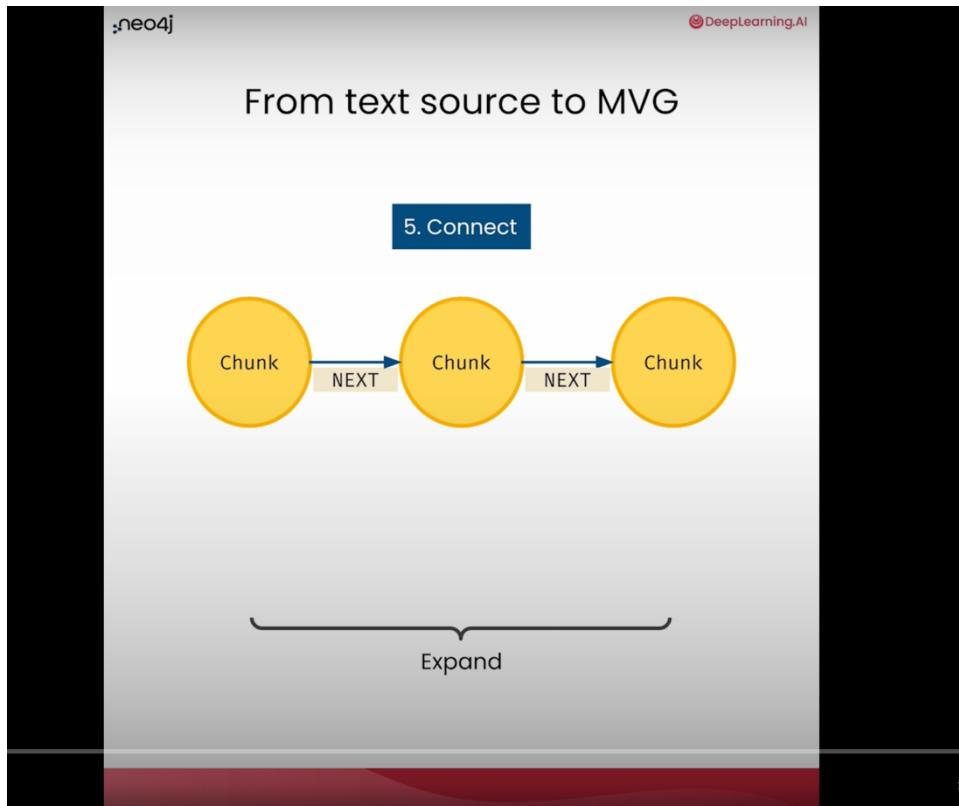
Start with a Minimum Viable Graph (MVG), then extract, enhance, expand, and repeat to grow the graph

Extract: identify interesting information

Enhance: supercharge the data

Expand: connect information to expand context

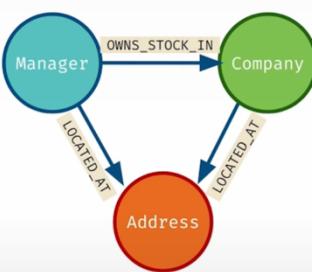




	Chunked Text Nodes	Form 10K Nodes	Companies	Management Firms
Source	Form 10K json	(:Chunk)	Form 13 CSV	Form 13 CSV
1. Extract	(:Chunk)	(:Form)	(:Company)	(:Manager)
2. Enhance	Vector embedding of text	Unique chunk IDs	Unique company cusip6	Full-text index of names
3. Expand	(Chunk) -[NEXT]→ (Chunk)	(Chunk) -[PART_OF]→ (Form)	(Company) -[FILED]→ (Form)	(Manager) -[OWNS_STOCK_IN]→ (Company)

Address nodes

Addresses	
Source	(:Company) & (:Manager) address strings
1. Extract	(:Address) with properties for City, State, Country
2. Enhance	Geospatial location index
3. Expand	(Manager Company) -[LOCATED_AT]-> (Address)

**Extract** (:Address) nodes

1. Extract full address strings
2. Geocode to obtain city, state, and country plus latitude and longitude

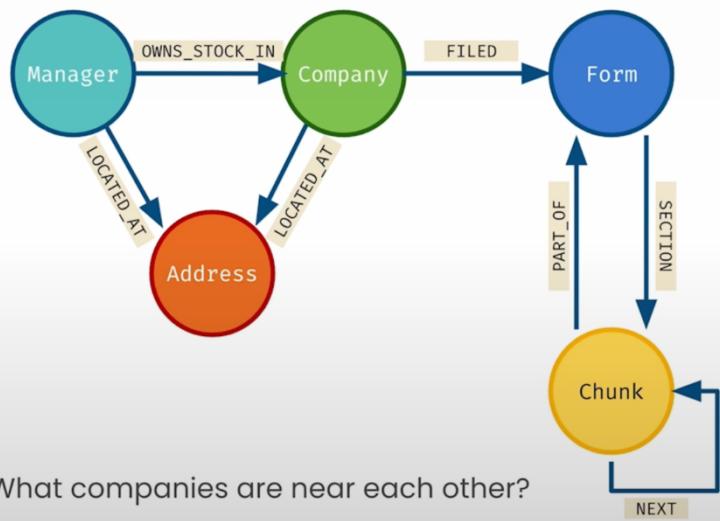
Enhance with geospatial index

1. Enable distance queries

Expand -[:LOCATED_AT]-> relationships

1. Connect (:Manager) and (:Company) nodes to (:Address)

Ask location questions



- Q:** What companies are near each other?
- Q:** How many investment firms are near the company they invested in?
- Q:** How many investment firms are in the same city as the company they invested in?

```
CYPHER_GENERATION_TEMPLATE = """Task:Generate Cypher statement to
query a graph database.
Instructions:
Use only the provided relationship types and properties in the
schema. Do not use any other relationship types or properties that
are not provided.
Schema:
{schema}
Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than
for you to construct a Cypher statement.
Do not include any text except the generated Cypher statement.
Examples: Here are a few examples of generated Cypher
statements for particular questions:

# What investment firms are in San Francisco?
MATCH (mgr:Manager)-[:LOCATED_AT]-(mgrAddress:Address)
    WHERE mgrAddress.city = 'San Francisco'
RETURN mgr.managerName
The question is:
{question}"""
```

```
CYPHER_GENERATION_PROMPT = PromptTemplate(
    input_variables=["schema", "question"],
    template=CYPHER_GENERATION_TEMPLATE
)
```

```
graphChain = GraphCypherQACodeChain.from_llm(
    ChatOpenAI(temperature=0),
    graph=kg,
    verbose=True,
    cypner_prompt=CYPHER_GENERATION_PROMPT,
```

```
cypherChain = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0),
    graph=kg,
    verbose=True,
    cypher_prompt=CYPHER_GENERATION_PROMPT,
)

def prettyCypherChain(question: str) -> str:
    response = cypherChain.run(question)
    print(textwrap.fill(response, 60))

prettyCypherChain("What investment firms are in San Francisco?")

> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (mgr:Manager)-[:LOCATED_AT]-(mgrAddress:Address)
WHERE mgrAddress.city = 'San Francisco'
RETURN mgr.managerName
Full Context:
[{'mgr.managerName': 'PARNASSUS INVESTMENTS, LLC'}, {'mgr.managerName': 'SKBA CAPITAL MANAGEMENT LLC'}, {'mgr.managerName': 'ROSENBLUM SILVERMAN SUTTON S F INC /CA'}, {'mgr.managerName': 'CHARLES SCHWAB INVESTMENT MANAGEMENT INC'}, {"mgr.managerName": "WELLS FARGO & COMPANY/MN"}, {"mgr.managerName": "Dodge & Cox"}, {"mgr.managerName": "Strait & Sound Wealth Management LLC"}, {"mgr.managerName": "Sonoma Private Wealth LLC"}, {"mgr.managerName": "Fund Management at Engine No. 1 LLC"}, {"mgr.managerName": "SELDON CAPITAL LP"}]

> Finished chain.
PARNASSUS INVESTMENTS, LLC, ROSENBLUM SILVERMAN SUTTON S F INC /CA, and Dodge & Cox are investment firms located in San Francisco.
```