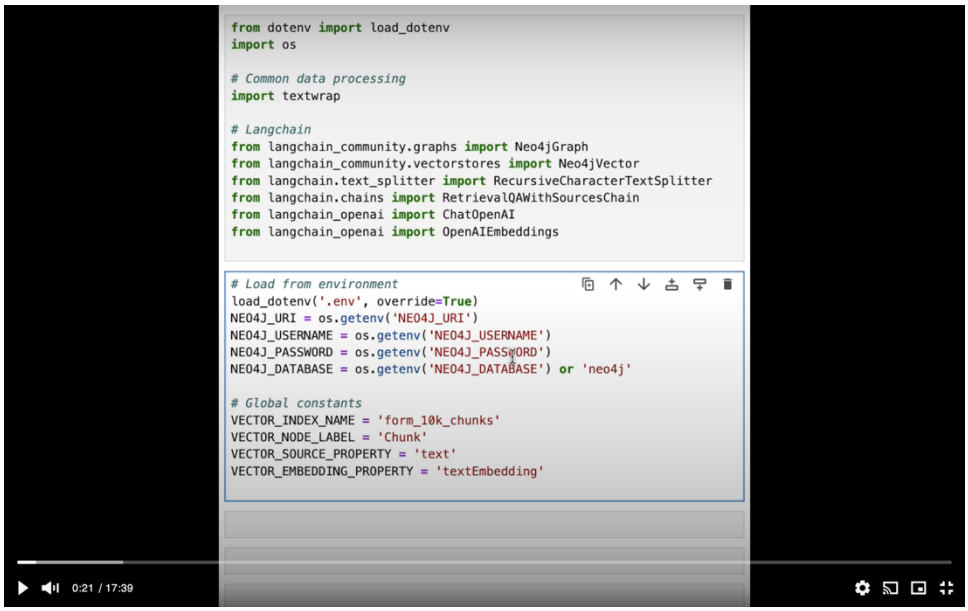
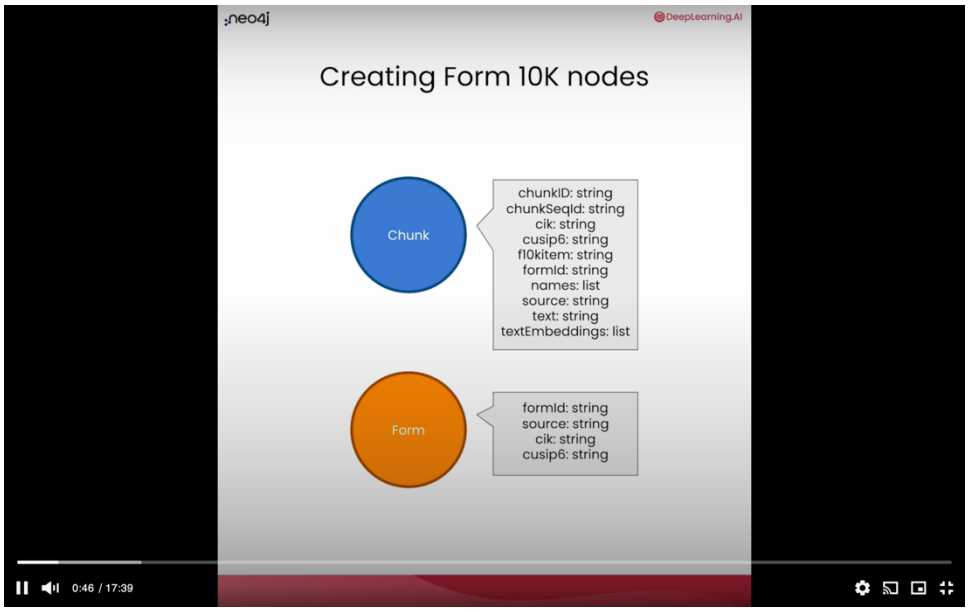
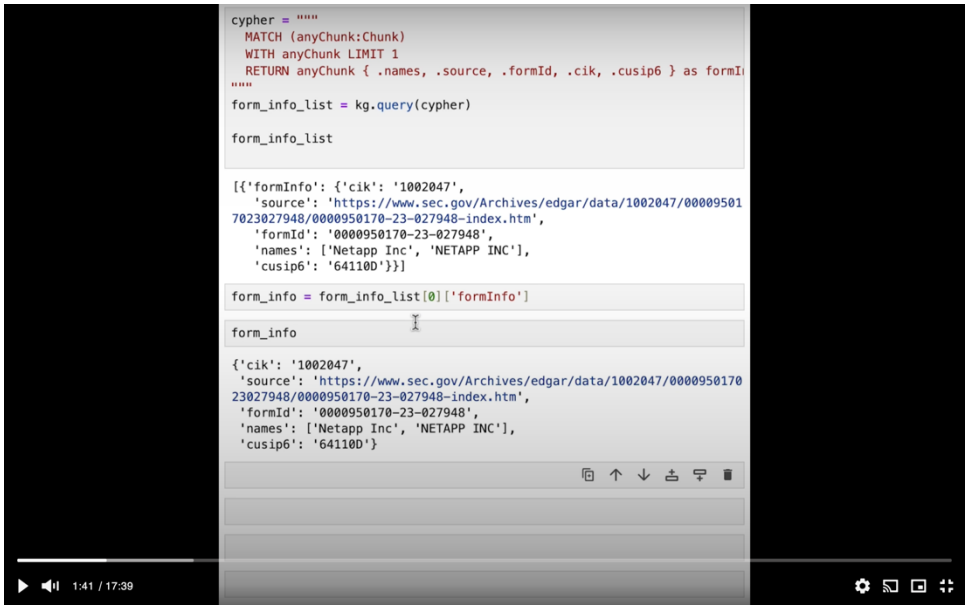
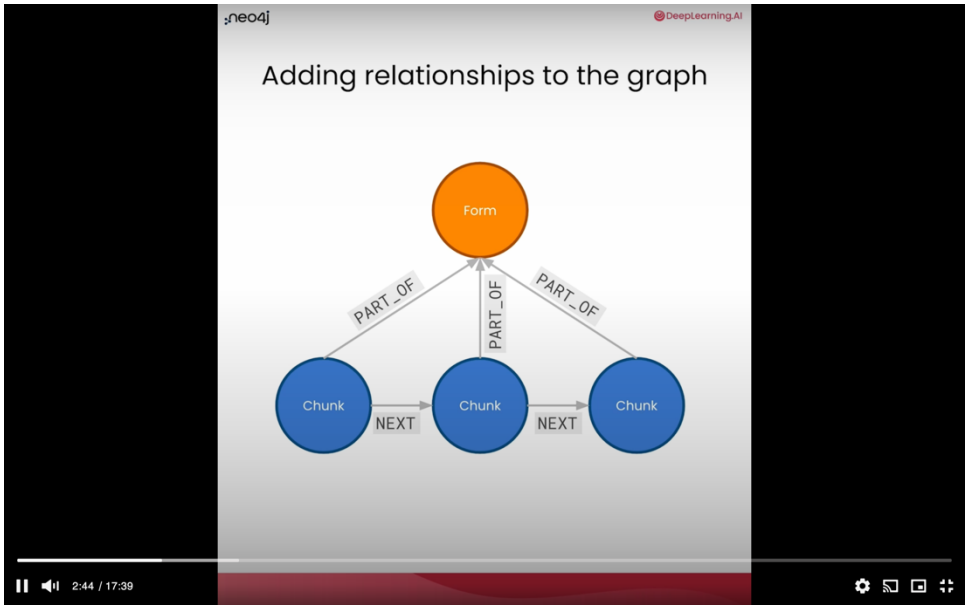
Lesson 5: Adding Relationships to the SEC Knowledge Graph





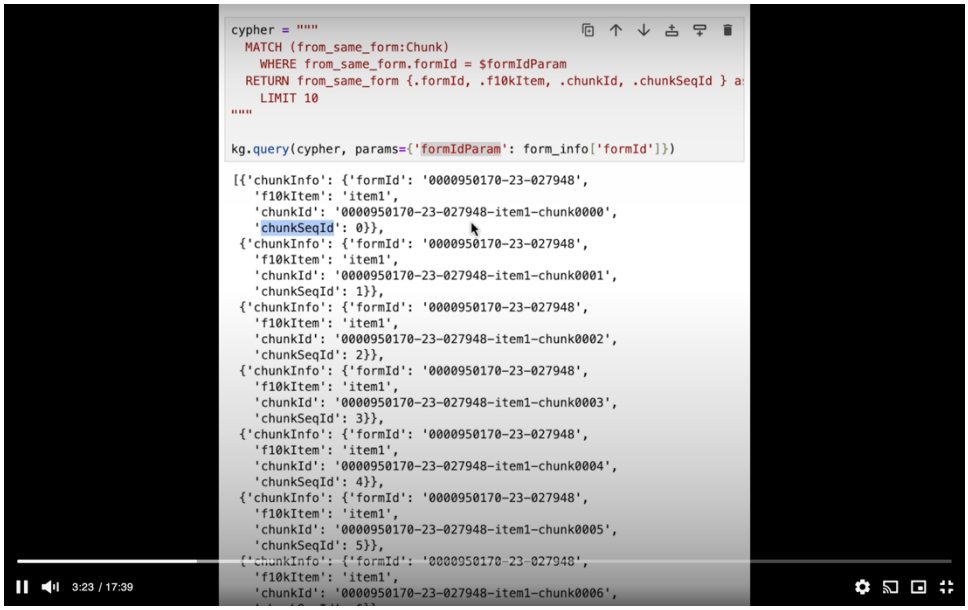

cypher = """
MATCH (anyChunk:Chunk)
WITH anyChunk LIMIT 1
RETURN anyChunk { .names, .source, .formId, .cik, .cusip6 } as formI
"""
form_info_list = kg.query(cypher)
form_info_list
[{'formInfo': {'cik': '1002047',
'source': 'https://www.sec.gov/Archives/edgar/data/1002047/00009501
7023027948/0000950170-23-027948-index.htm',
'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



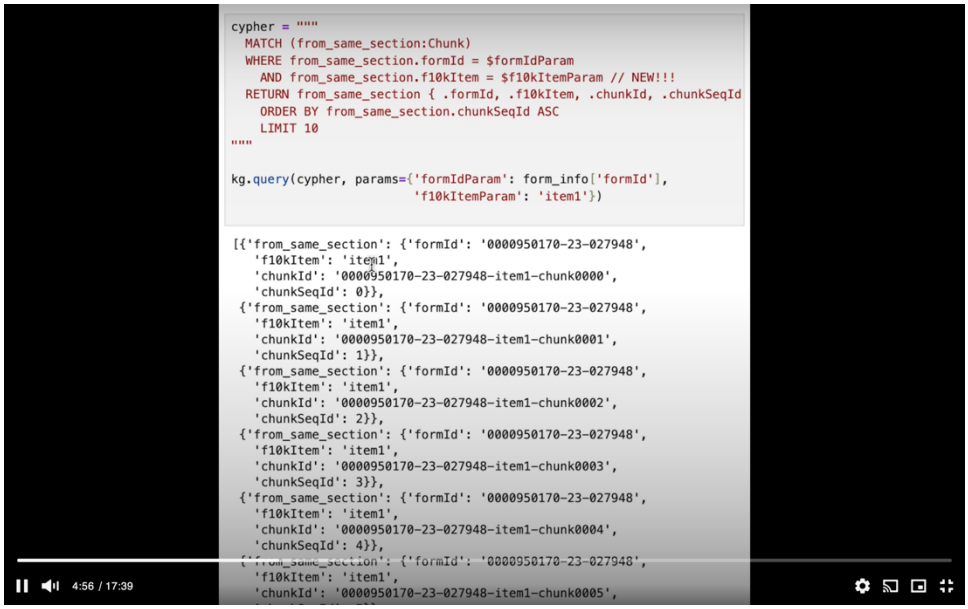
So chunks are in linkedlist.

Now lets query all chunks from a particular form id:

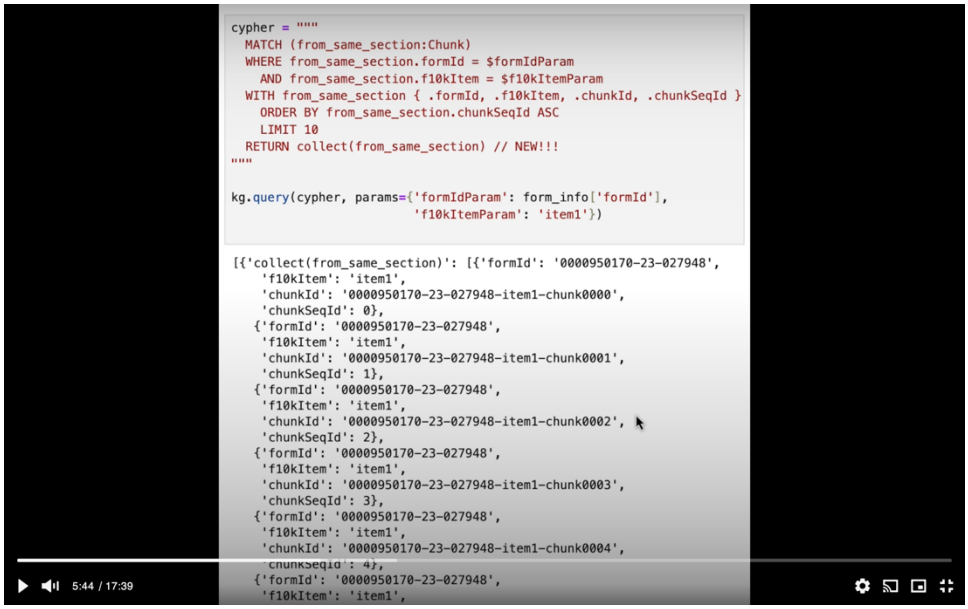


now we want chuckseqid in ascending order, as linkedlist, so order by.

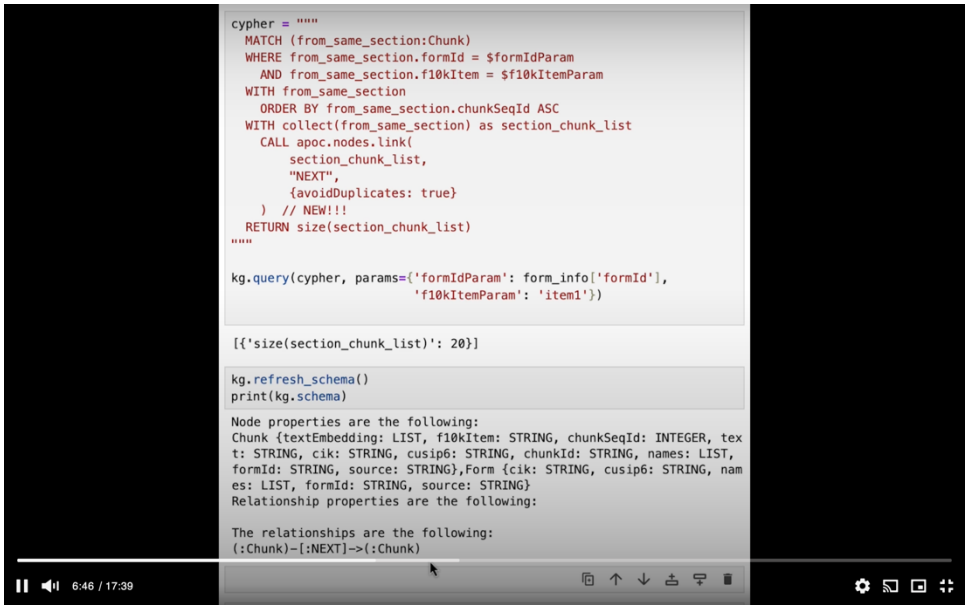
And also from same section.
So. ascending chunks id, within each section,



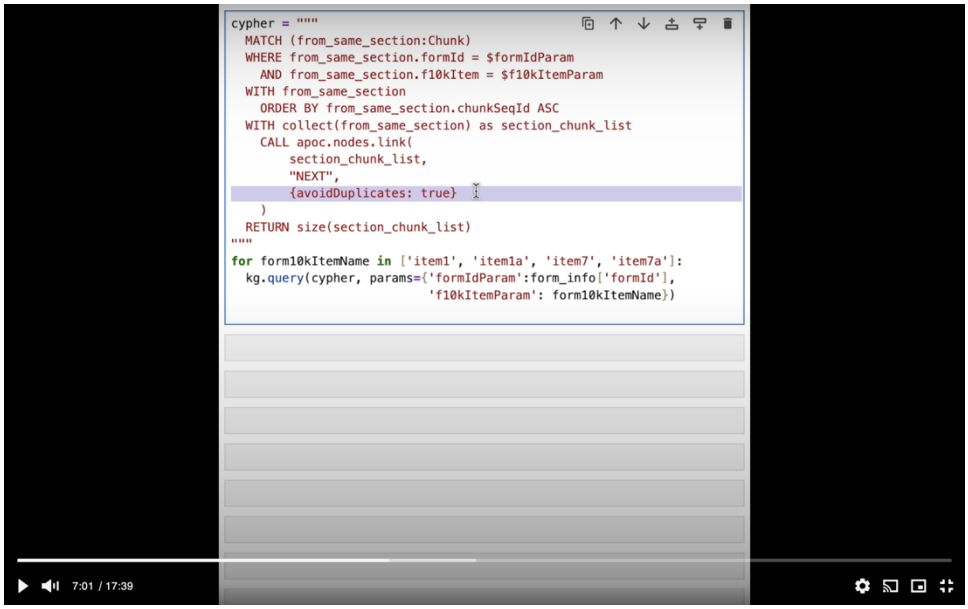
We use COLLECT to get the output as a list. With one row.



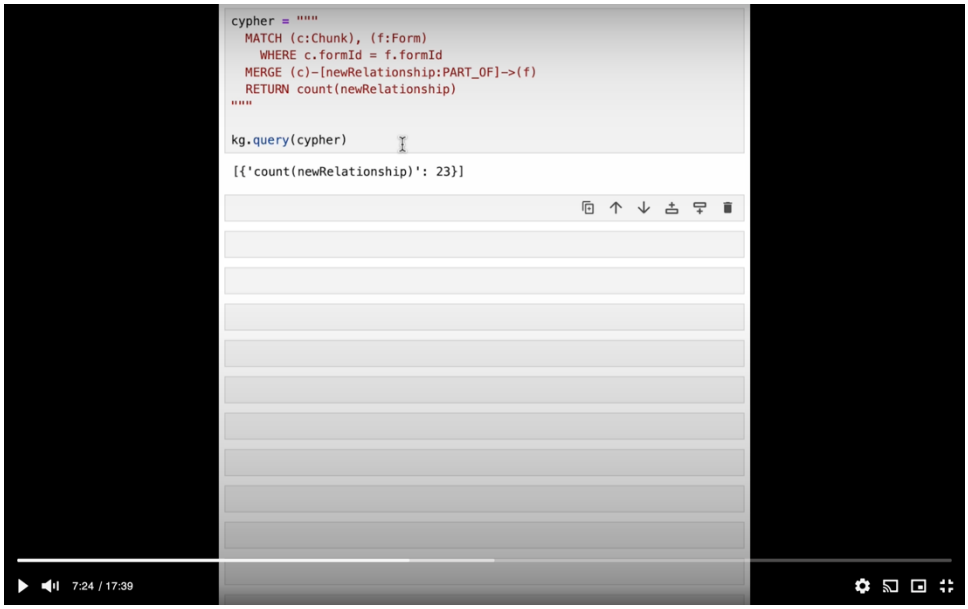
Now we add relationship for these chunks:



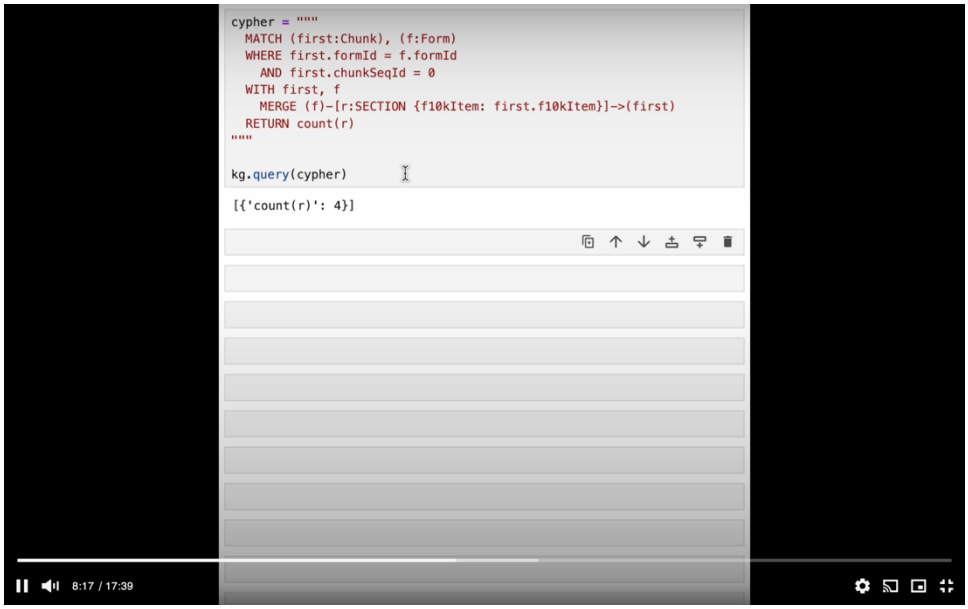
Now we do that für all items:



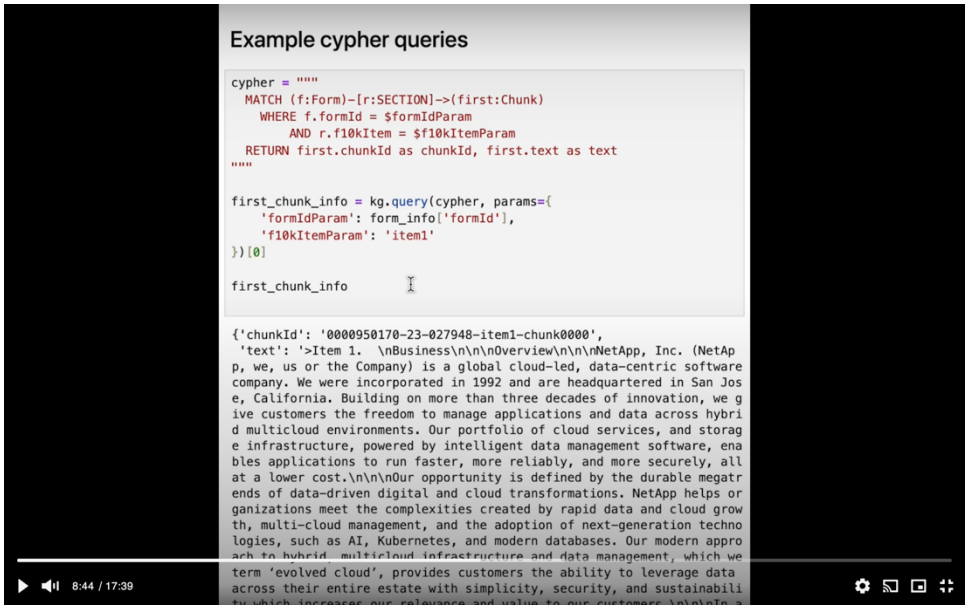
Relationship between chunks and form it belongs to:



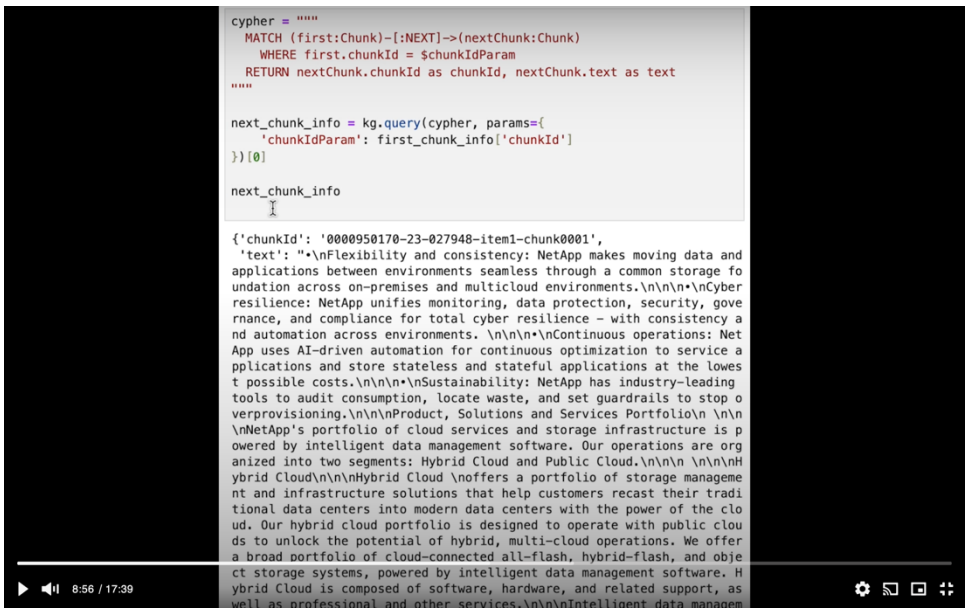
Linking first chunk to forms. To identify diff sections_



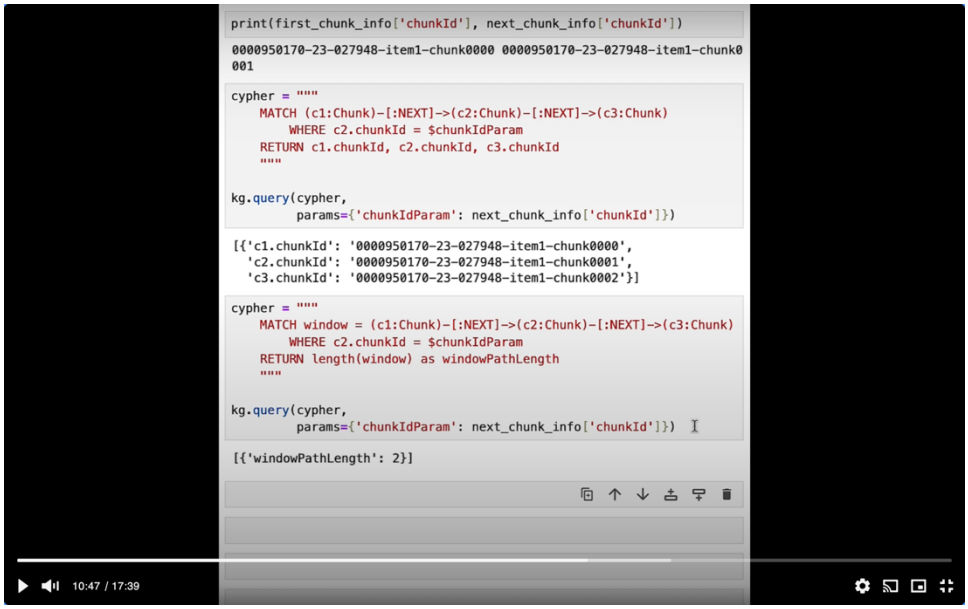
Query-_:



With chunk_id: find next chunk t it:

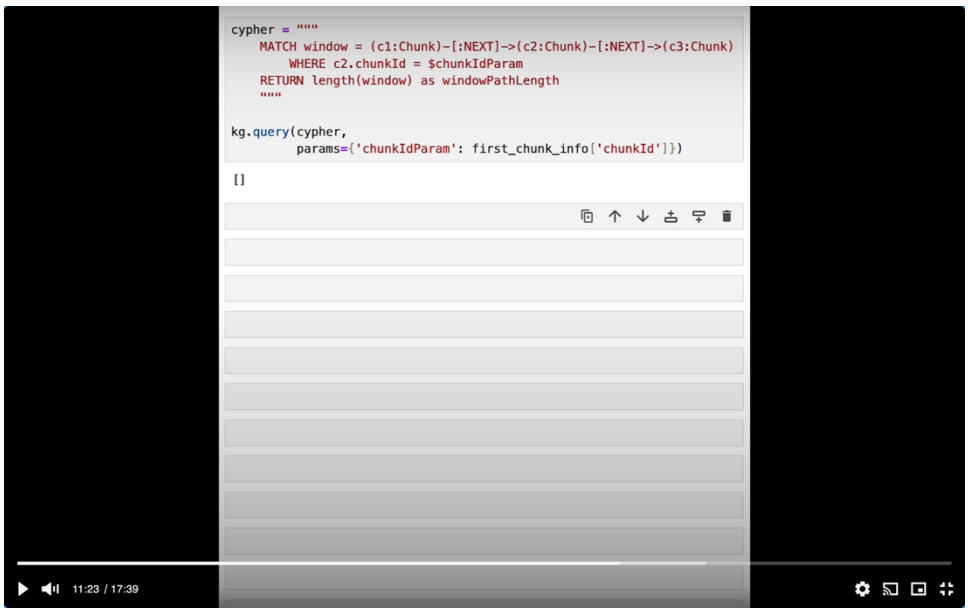


Get one before and one later chunks, and also path, or window::

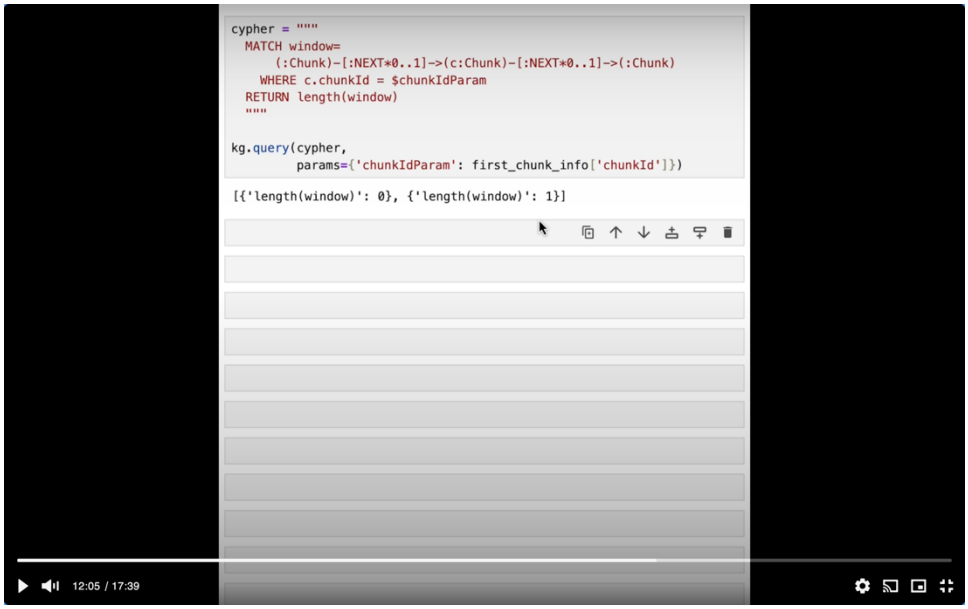


V

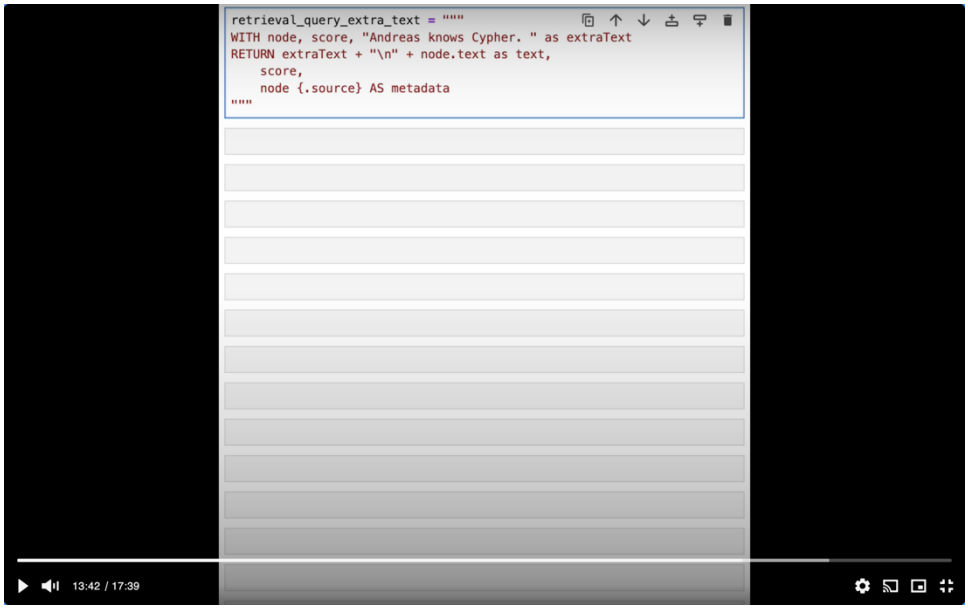
if we try to match first chunk, and ask for one before chunk, that it wont return anything:



How to solve it, range
Min 0, max 1 match


cypher = """
MATCH window=
(:Chunk)-[:NEXT*0..1]->(c:Chunk)-[:NEXT*0..1]->(:Chunk)
WHERE c.chunkId = $chunkIdParam
RETURN length(window)
"""
kg.query(cypher,
params={'chunkIdParam': first_chunk_info['chunkId']})
[{'length(window)': 0}, {'length(window)': 1}]
12:05 / 17:39


cypher = """
MATCH window=
(: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}]
12:47 / 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
"""
13:42 / 17:39

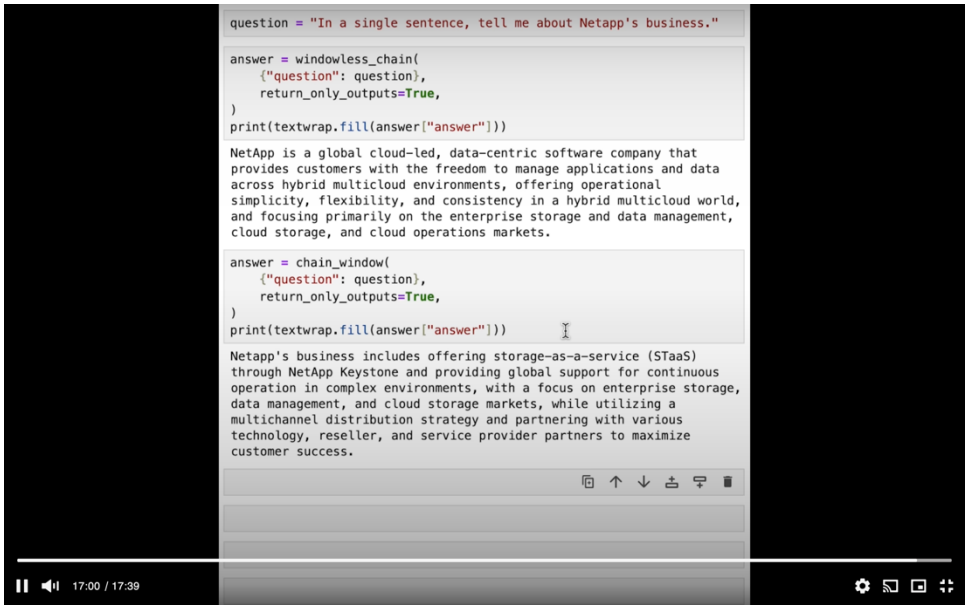

retrieval_query_extra_text = """
WITH node, score, "Andreas knows Cypher. " as extraText
RETURN extraText + "\n" + node.text as text,
score,
node {.source} AS metadata
"""
vector_store_extra_text = Neo4jVector.from_existing_index(
embedding=OpenAIEmbeddings(),
url=NEO4J_URI,
username=NEO4J_USERNAME,
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
14:02 / 17:39

Now lets use window:

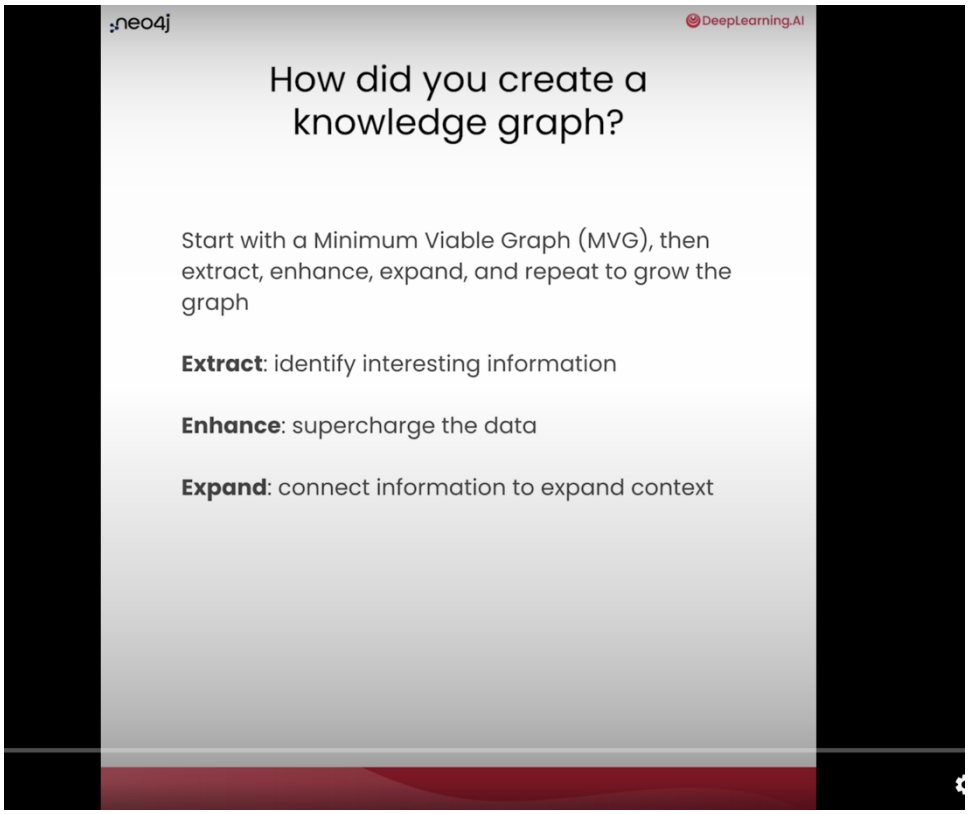
Lets create onne chain first without window to compare:

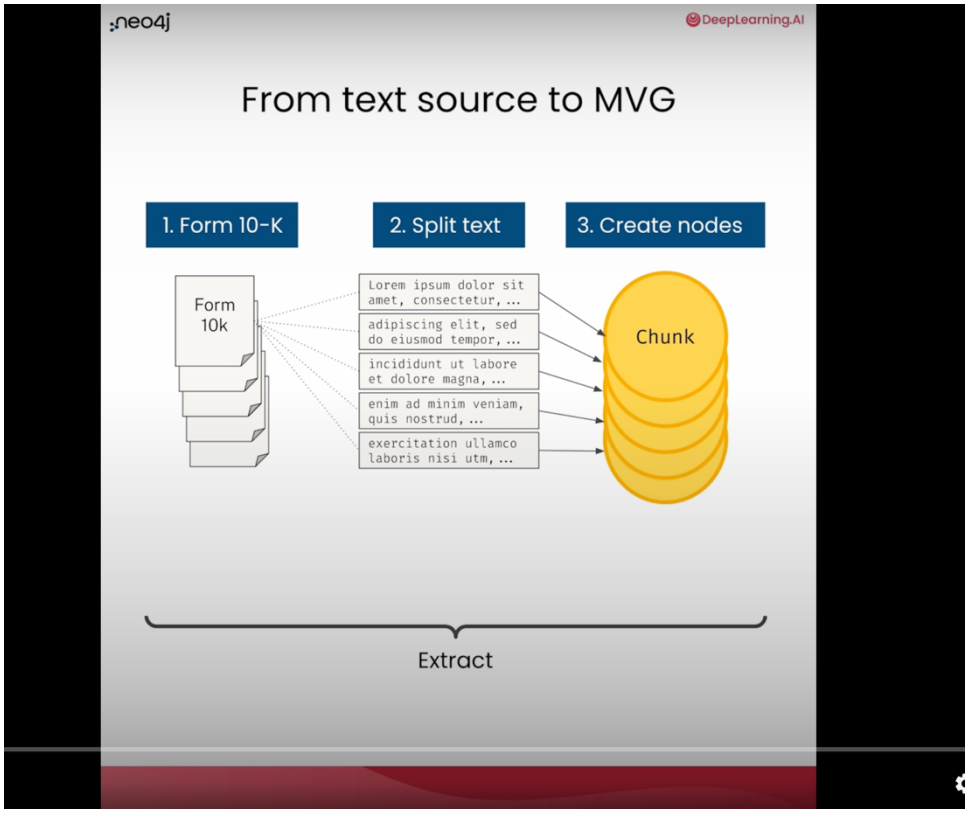
With window:

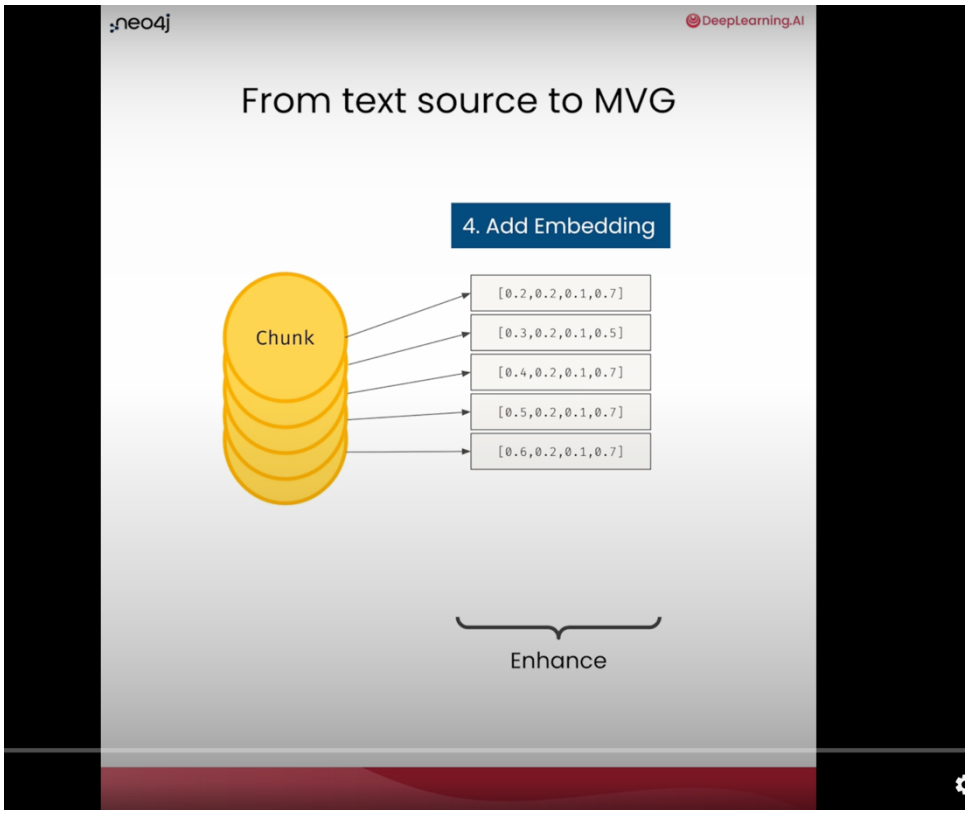
Example:

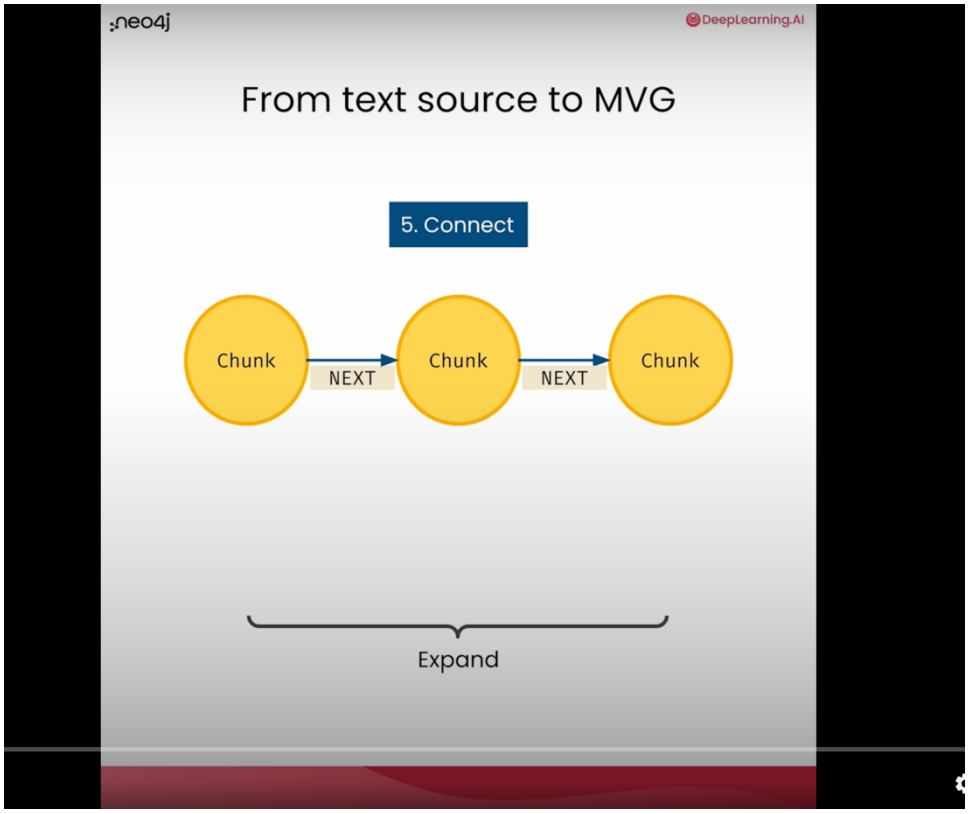


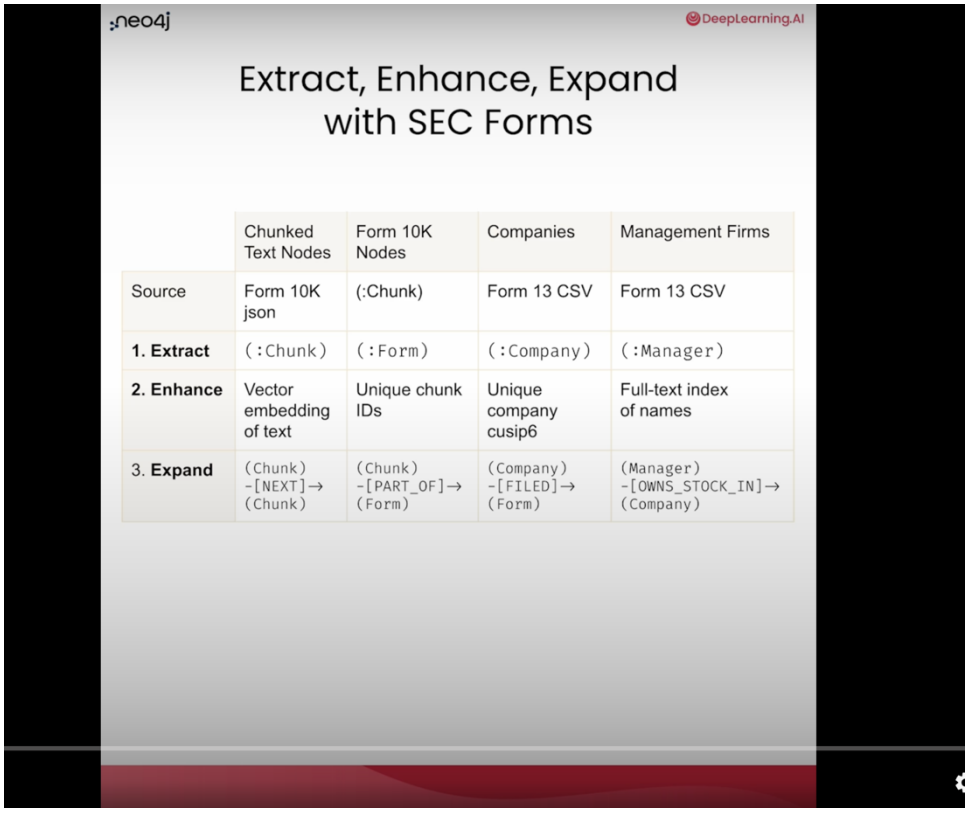
Final part:


:neo4j
DeepLearning.AI
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


:neo4j
DeepLearning.AI
From text source to MVG
1. Form 10-K
2. Split text
3. Create nodes
Form 10k
Lorem ipsum dolor sit amet, consectetur, ...
adipiscing elit, sed do eiusmod tempor, ...
incididunt ut labore et dolore magna, ...
enim ad minim veniam, quis nostrud, ...
exercitation ullamco laboris nisi utm, ...
Chunk
Extract


:neo4j
DeepLearning.AI
From text source to MVG
4. Add Embedding
Chunk
[0.2,0.2,0.1,0.7]
[0.3,0.2,0.1,0.5]
[0.4,0.2,0.1,0.7]
[0.5,0.2,0.1,0.7]
[0.6,0.2,0.1,0.7]
Enhance


neo4j
DeepLearning.AI
From text source to MVG
5. Connect
Chunk
NEXT
Chunk
NEXT
Chunk
Expand


neo4j
DeepLearning.AI
Extract, Enhance, Expand with SEC Forms
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)

neo4j

DeepLearning.AI

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

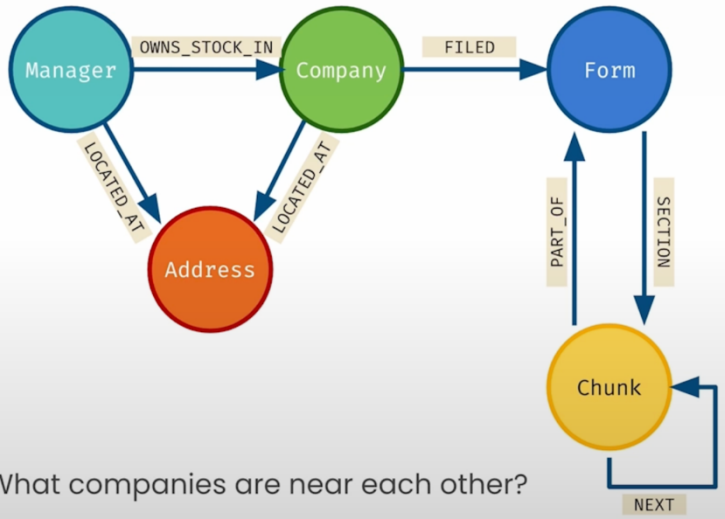
1. Enable distance queries

Expand -[:LOCATED_AT]-> relationships

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

:neo4j DeepLearning.AI

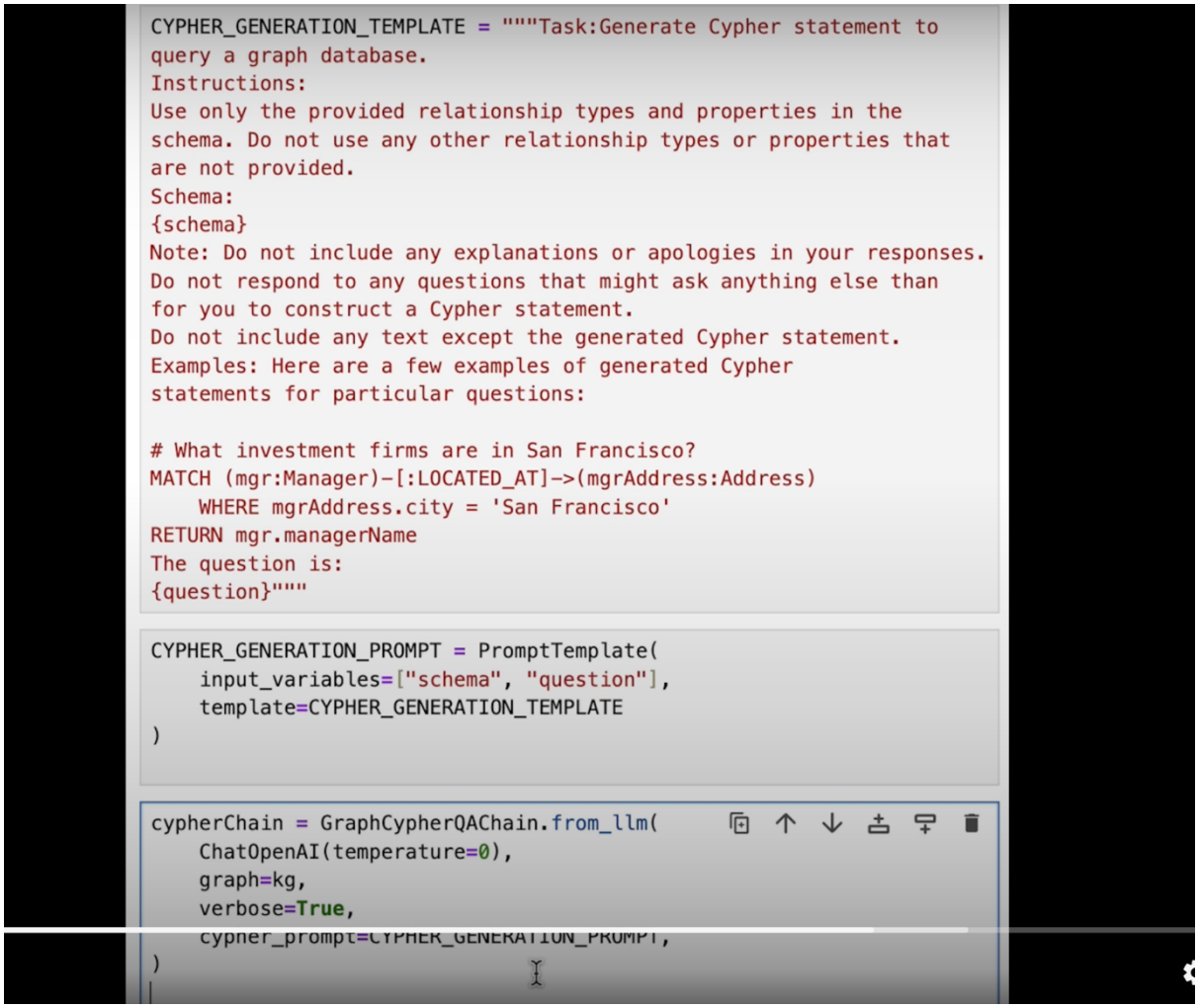
Ask location questions



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
)
cypherChain = GraphCypherQAChain.from_llm(
ChatOpenAI(temperature=0),
graph=kg,
verbose=True,
cypher_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.managerNam
e': 'SKBA CAPITAL MANAGEMENT LLC'}, {'mgr.managerName': 'ROSENBLUM SI
LVERMAN SUTTON S F INC /CA'}, {'mgr.managerName': 'CHARLES SCHWAB INV
ESTMENT 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 LL
C'}, {'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.
```