

[Open in app](#)

Search Medium

Get unlimited access to all of Medium for less than \$1/week. [Become a member](#)

LangChain has added Cypher Search

With the LangChain library, you can conveniently generate Cypher queries, enabling an efficient retrieval of information from Neo4j.



tds

Tomaz Bratanić · [Follow](#)

Published in Towards Data Science

8 min read · May 25



Listen



Share



More



Image generated by Midjourney paid subscription. [Midjourney ToS](#).

If you have developed or plan to implement any solution that uses Large Language Models, you have most likely heard of the [LangChain library](#). LangChain library is the most widely known Python library used to develop applications that use LLMs in one or another capabilities. It is designed to be modular, allowing us to use any LLM in any available modules, such as chains, tools, memory, or agents.

A month ago, I spent a week researching and implementing a solution allowing

anyone to retrieve information from [Neo4j](#) directly from the LangChain library and use it in their LLM applications. I learned quite a lot about the internals of LangChain library and wrote up my experience in a [blog post](#).

A colleague of mine showed me a LangChain feature request, where the user requested that my work of having the option to retrieve information from the Neo4j database would be added as a module directly to the LangChain library so that no additional code or external modules would be needed to integrate Neo4j into LangChain applications. Since I was already familiar with LangChain internals, I decided to try and implement Cypher searching capabilities myself. I spent a weekend researching and coding the solution and ensuring it would conform to the contribution standards for it to be added to the library. Luckily, the maintainers of LangChain are very responsive and open to new ideas, and the Cypher Search has been added in the latest release of the LangChain library. Thanks to [Harrison Chase](#) for maintaining such a great library and also being very responsive to new ideas.

In this blog post, I will show you how you can use the newly added Cypher Search in the LangChain library to retrieve information from a Neo4j database.

The code is available on [GitHub](#).

What is a knowledge graph

LangChain has already integrations with Vector and SQL databases, so why do we need an integration with a Graph Database like Neo4j?



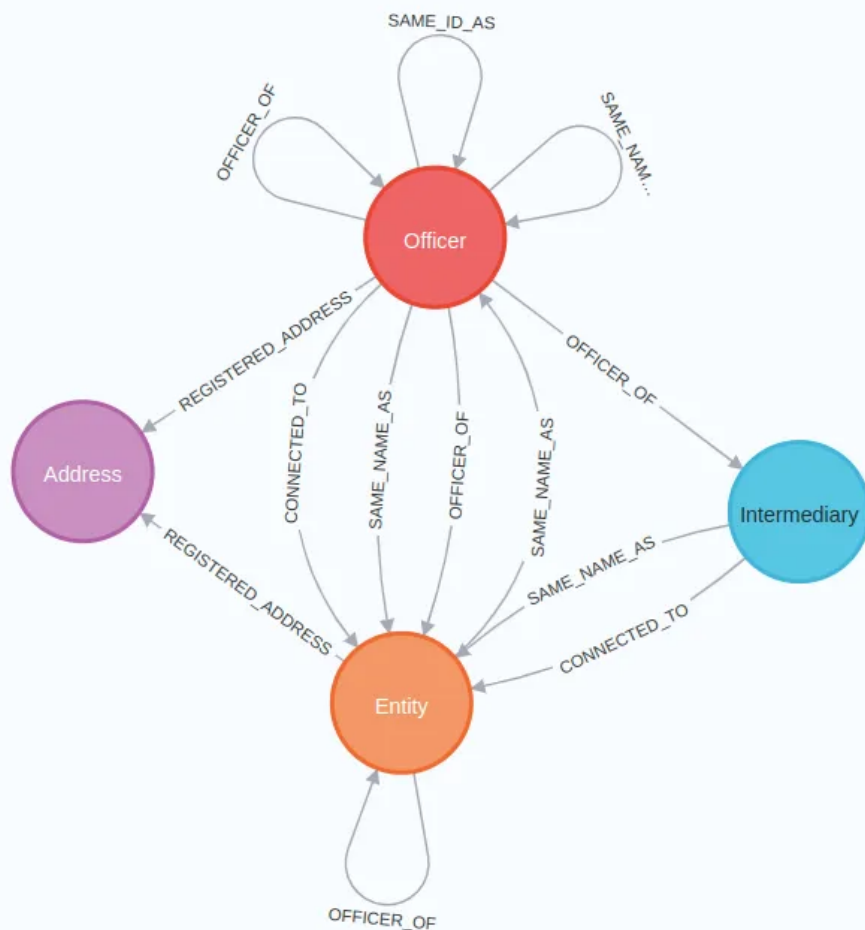
The power of graph databases truly shines when dealing with complex systems where interdependencies and interactions are vital in understanding the system.

They enable us to see beyond individual data points and delve into the intricate relationships that define their context. This provides a deeper, more holistic view of the data, facilitating better decision-making and knowledge discovery.

Setting up Neo4j environment

If you have an existing Neo4j database, you can use it to try the newly added Cypher Search. The Cypher Search module uses graph schema information to generate Cypher statements, meaning you can plug it into any Neo4j database.

If you don't have any Neo4j database yet, you can use [Neo4j Sandbox](#), which offers a free cloud instance of a Neo4j database. You need to register and instantiate any of the available pre-populated databases. I will be using the [**ICIJ Paradise Papers dataset**](#) in this blog post, but you can use any other if you want. The dataset has been made available by [International Consortium of Investigative Journalists](#) as part of their [Offshore Leaks Database](#).



Paradise Papers dataset graph schema. Image by the author.

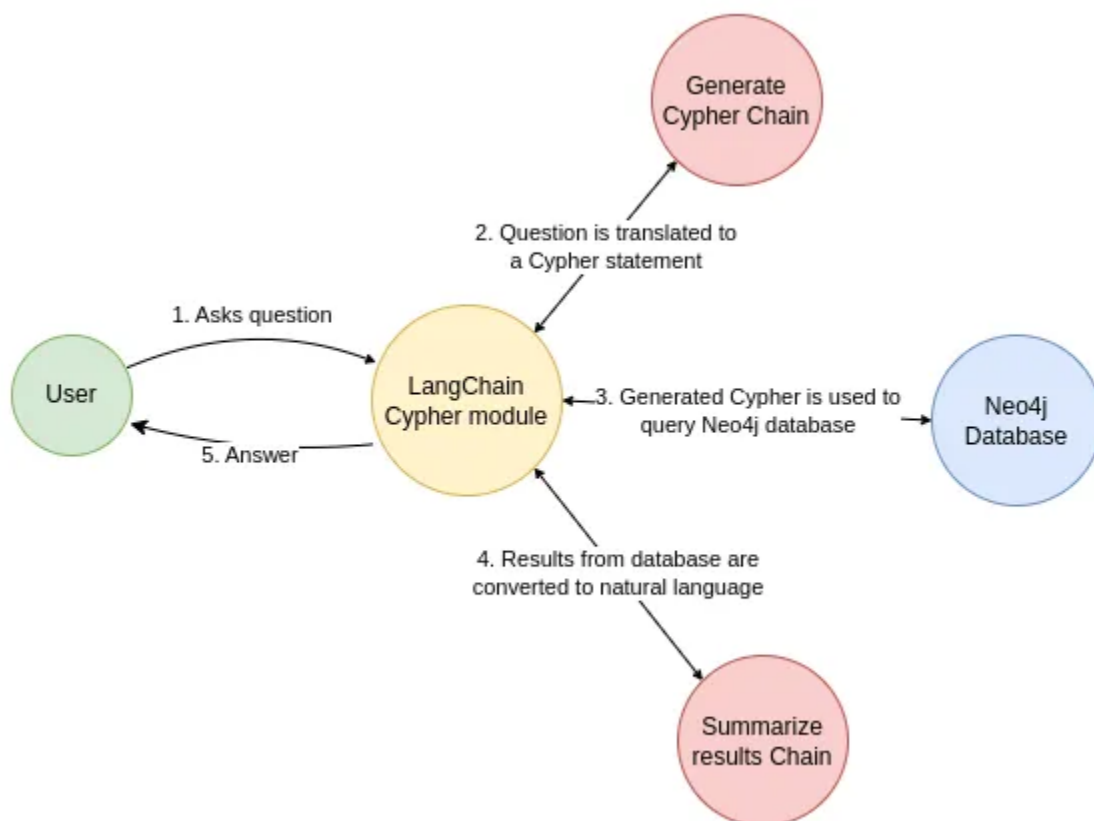
The graph contains four types of nodes:

- **Entity** - The offshore legal entity. This could be a company, trust, foundation, or other legal entity created in a low-tax jurisdiction.
- **Officer** - A person or company who plays a role in an offshore entity, such as beneficiary, director, or shareholder. The relationships shown in the diagram are just a sample of all the existing ones.
- **Intermediary** - A go-between for someone seeking an offshore corporation and an offshore service provider — usually a law-firm or a middleman that asks an offshore service provider to create an offshore firm.

- **Address** - The registered address as it appears in the original databases obtained by ICIJ.

Knowledge Graph Cypher Search

The name Cypher Search comes from Cypher, which is a query language used to interact with graph databases like Neo4j.



Knowledge graph Cypher chain workflow. Image by the author.

In order to allow LangChain to retrieve information from graph databases, I implemented a module that can convert the natural language to a Cypher statement, use it to retrieve data from Neo4j and return the retrieved information to the user in a natural language form. This two-way conversion process between natural language and database language not only enhances the overall accessibility of data retrieval but also greatly improves the user experience.

The beauty of the LangChain library is in its simplicity. We only need a couple of lines of code and we can retrieve information from Neo4j using natural language.

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import GraphCypherQAChain
from langchain.graphs import Neo4jGraph

graph = Neo4jGraph(
    url="bolt://54.172.172.36:7687",
    username="neo4j",
    password="steel-foreheads-examples"
)

chain = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0), graph=graph, verbose=True,
)
```

Here, we are using the **gpt-3.5-turbo** model from OpenAI to generate Cypher statements. The Cypher statements are generated based on the graph schema, which means that, in theory, you can plug the Cypher chain into any Neo4j instance, and it should be able to answer natural language answers. Unfortunately, I haven't yet tested other LLM providers in their ability to generate Cypher statements since I don't have access to any of them. Still, I would love to hear your evaluation of other LLMs generating Cypher statements if you will give it a go. Of course, if you want to break the dependency on LLM cloud providers, you can always fine-tune an open-source LLM to generate Cypher statements.

Let's start with a simple test.

```
chain.run("""
Which intermediary is connected to most entites?
""")
```

Results


```

> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (i:Intermediary)-[:CONNECTED_TO]->(e:Entity)
RETURN i.name, COUNT(e) AS num_entities
ORDER BY num_entities DESC
LIMIT 1
Full Context:
[{'i.name': 'Group - SUN Capital Partner Group', 'num_entities': 115}]

> Finished chain.
'The intermediary connected to the most entities is the SUN Capital Partner Group with 115 entities.'

```

Generated answer. Image by the author.

We can observe the generated Cypher statement and the retrieved information from Neo4j used to form the answer. That is as easy a setup as it gets. Let's move on to the next example.

```

chain.run("""
Who are the officers of ZZZ-MILI COMPANY LTD.?
""")

```

Results

```

> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (e:Entity {name: 'ZZZ-MILI COMPANY LTD.'})<-[:OFFICER_OF]-(o:Officer) RETURN o.name
Full Context:
[{'o.name': 'Whitson - Iva Marie'}, {'o.name': 'Whitson - Claud S.'}, {'o.name': 'Whitson - Claud S.'}, {'o.name': 'Whitson - Claud S.'}]

> Finished chain.
'The officers of ZZZ-MILI COMPANY LTD. are Whitson - Iva Marie, Whitson - Claud S., FINSBURY NOMINEES LTD, EURO NOMINEES LTD., EURO SECURITIES LTD., and COMPANY DIRECTORS LTD.'

```

Generated answer. Image by the author.

Since we are using a graph, let's construct a question that would utilize the power of graph databases.

```

chain.run("""
How are entities SOUTHWEST LAND DEVELOPMENT LTD. and

```

```

Dragon Capital Markets Limited related?
""")
# Generated Cypher statement
# MATCH (e1:Entity {name: 'SOUTHWEST LAND DEVELOPMENT LTD.'})-
# [:CONNECTED_TO|OFFICER_OF|INTERMEDIARY_OF|SAME_NAME_AS*]-(e2:Entity {name: 'D

```

The generated Cypher statement looks fine at first glance. However, there is a problem as the Cypher statements used the variable-length path finding syntax and also treated relationships as undirected. As a result, this type of query is highly unoptimized and would explode in the number of rows.

The nice thing about gpt-3.5-turbo is that it follows hints and instructions we drop in the input. For example, we can ask it to find only the shortest path.

```

chain.run("""
How are entities SOUTHWEST LAND DEVELOPMENT LTD. and
Dragon Capital Markets Limited connected?
Find a shortest path.
""")

```

> Entering new GraphCypherQAChain chain...

Generated Cypher:

```

MATCH (e1:Entity {name: 'SOUTHWEST LAND DEVELOPMENT LTD.'}), (e2:Entity {name: 'Dragon Capital Marke
MATCH p=shortestPath((e1)-[*]-(e2))
RETURN p

```

Full Context:

```

[{'p': [{'sourceID': 'Paradise Papers - Appleby', 'jurisdiction': 'KY', 'service_provider': 'Appleby

```

> Finished chain.

```

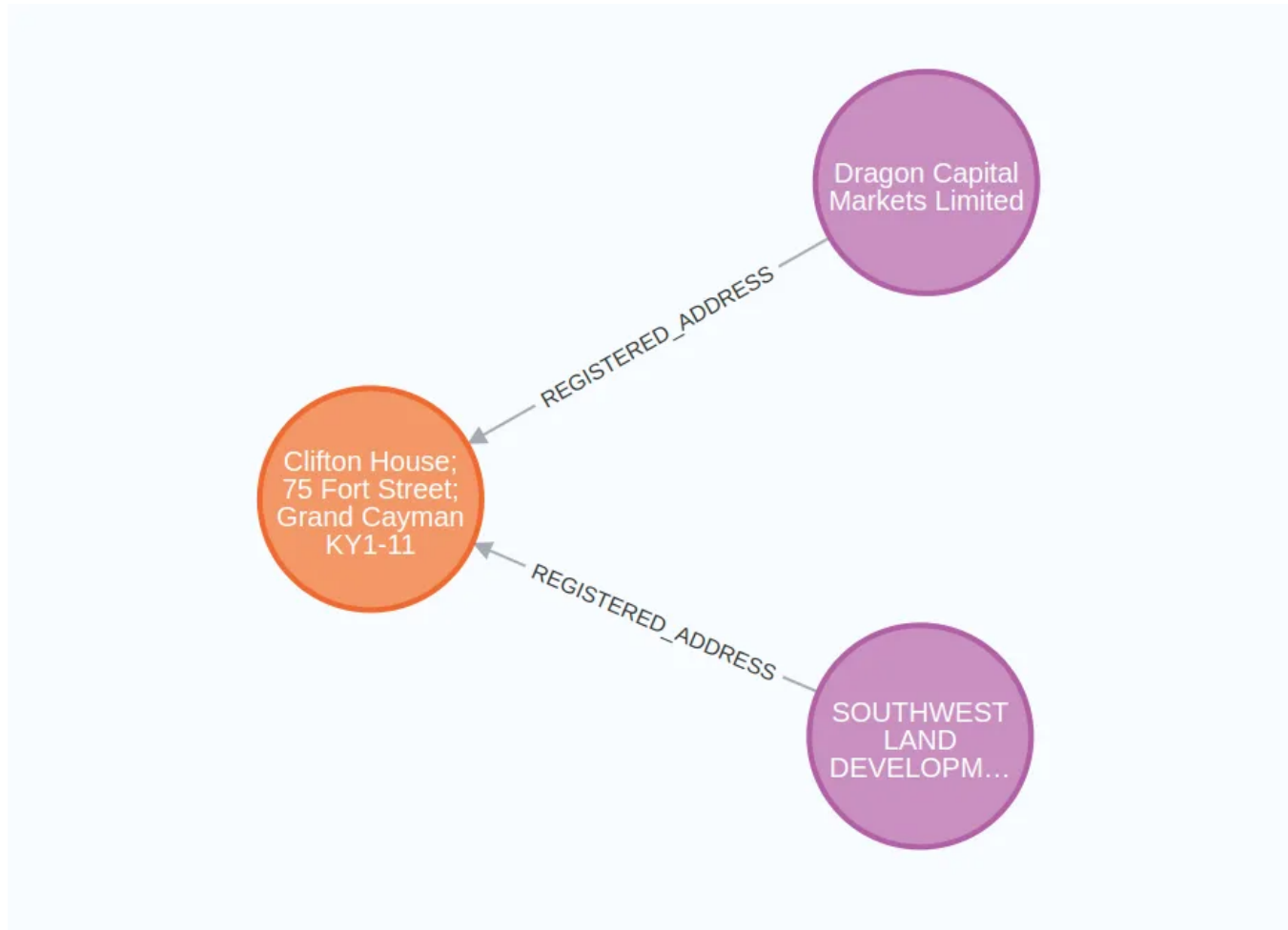
'SOUTHWEST LAND DEVELOPMENT LTD. and Dragon Capital Markets Limited are connected through their com
mon service provider, Appleby, and their shared jurisdiction of Cayman Islands. A possible shortest
path between the two entities could be: SOUTHWEST LAND DEVELOPMENT LTD. -> Appleby -> Cayman Island
s -> Dragon Capital Markets Limited.'

```

Generated answer. Image by the author.

Now that we dropped a hint that only the shortest path should be retrieved, we don't run into cardinality explosion troubles anymore. However, one thing I noticed is that the LLM sometimes doesn't provide the best results if a path object is returned. The generated Cypher statement returns the following visualization in Neo4j

Browser.



Graph visualization of the answer. Image by the author.

The generated natural language response didn't really mention that the two companies are registered at the same address, but made it its own shortest path based on the node properties. However, we can also fix that by instructing the model what information to use.

```
chain.run("""
How are entities SOUTHWEST LAND DEVELOPMENT LTD. and Dragon Capital Markets Lim
Find a shortest path.
Return only name properties of nodes and relationship types
""")
```

Results

> Entering new GraphCypherQAChain chain...

Generated Cypher:

```
MATCH path = shortestPath((e1:Entity {name: 'SOUTHWEST LAND DEVELOPMENT LTD.'})-[*]-(e2:Entity {name: 'Dragon Capital Markets Limited'})
RETURN [n IN nodes(path) | n.name] AS Names, [r IN relationships(path) | type(r)] AS Relationships
```

Full Context:

```
[{'Names': ['SOUTHWEST LAND DEVELOPMENT LTD.', 'Clifton House; 75 Fort Street; Grand Cayman KY1-1108; Cayman Islands (REGISTERED_
```

> Finished chain.

'The entities SOUTHWEST LAND DEVELOPMENT LTD. and Dragon Capital Markets Limited are connected through a registered address. The shortest path between them is: SOUTHWEST LAND DEVELOPMENT LTD. (REGISTERED ADDRESS) -> Clifton House; 75 Fort Street; Grand Cayman KY1-1108; Cayman Islands (REGISTERED_

Neo4j

Langchain

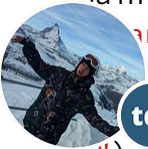
Graph

Gpt 3

Gpt 4

Generated answer. Image by the author.

Now we can a better response and more appropriate response. The more hints you drop to an LLM, the better results you can expect. For example, you can also instruct it which relationships it can traverse.



chain.run("""

are entities SOUTHWEST LAND DEVELOPMENT LTD. and Dragon Capital Markets Lim

a shortest path and use only officer, intermediary, and connected relation

only name properties of nodes and relationship types

""")

Follow

Written by Tomaz Bratanić

3.5K Followers · Writer for Towards Data Science

Results

Data explorer. Turn everything into a graph. Author of Graph algorithms for Data Science at Manning

publication: <http://mg.bz/GGVN>

> Entering new GraphCypherQAChain chain...

Generated Cypher:

```
MATCH (e1:Entity {name: 'SOUTHWEST LAND DEVELOPMENT LTD.'}), (e2:Entity {name: 'Dragon Capital Mark
MATCH p=shortestPath((e1)-[:OFFICER_OF|INTERMEDIARY_OF|CONNECTED_TO*]-(e2))
RETURN [n IN nodes(p) | n.name] AS Names, [r IN relationships(p) | type(r)] AS Relationships
```

Full Context:

```
[{'Names': ['SOUTHWEST LAND DEVELOPMENT LTD.', 'Appleby Trust (Cayman) Ltd.', 'Dragon Capital Clean
```

> Finished chain.

'SOUTHWEST LAND DEVELOPMENT LTD. is an intermediary of Appleby Trust (Cayman) Ltd., which is an intermediary of Dragon Capital Clean Development Investments Ltd. Dragon Capital Clean Development Investments Ltd. is connected to Group - Dragon Capital, which is connected to Dragon Capital Markets Limited. Therefore, SOUTHWEST LAND DEVELOPMENT LTD. and Dragon Capital Markets Limited are connected through a chain of intermediaries and connected entities. The shortest path between them involves two intermediary relationships and two connected relationships.'

Generated answer. Image by the author.