

```
= The Cypher QA Chain
:order: 1
:type: lesson
:disable-cache: true
```

Language models and vector indexes are good at querying unstructured data. Although, as you have seen, responses are not always correct, and when data is structured, it is often easier to query it directly.

LLMs are good at writing Cypher queries when given good information, such as:

- * The schema of the graph
- * Context about the question to be answered
- * Examples of questions and appropriate Cypher queries

In this lesson, you will learn how to use a language model to generate Cypher queries to query a Neo4j graph database.

```
== Generating Cypher
```

Langchain includes the [link:https://api.python.langchain.com/en/latest/_modules/langchain/chains/graph_qa/cypher.html#GraphCypherQAChain](https://api.python.langchain.com/en/latest/_modules/langchain/chains/graph_qa/cypher.html#GraphCypherQAChain) chain that can interact with a Neo4j graph database. It uses a language model to generate Cypher queries and then uses the graph to answer the question.

``GraphCypherQAChain`` chain requires the following:

- * An LLM (``llm``) for generating Cypher queries
- * A graph database connection (``graph``) for answering the queries
- * A prompt template (``cypher_prompt``) to give the LLM the schema and question
- * An appropriate question which relates to the schema and data in the graph

The program below will generate a Cypher query based on the schema in the graph database and the question.

Review the code and predict what will happen when you run it.

```
[source,python]
----
include::code/cypher-gen.py[tag=**]
----
```

[NOTE]

Before running the program, you must update the ``openai_api_key`` and the ``graph`` connection details.

```
[%collapsible]
.Click to reveal your Sandbox connection details
====
```

```
Connection URL:: [copy]#bolt://{sandbox-ip}:{sandbox-boltPort}#
Username:: [copy]#{sandbox-username}#
Password:: [copy]#{sandbox-password}#
====
```

When you run the program, you should see the Cypher generated from the question and the data it returned. Something similar to:

```
Generated Cypher:
MATCH (m:Movie {title: "Toy Story"})
RETURN m.plot
Full Context:
[{'m.plot': "A cowboy doll is profoundly threatened and jealous when a
new spaceman
figure supplants him as top toy in a boy's room."}]
```

The LLM used the database schema to generate an `_appropriate_` Cypher query.

Langchain then executed the query against the graph database, and the result returned.

== Breaking Down the Program

Reviewing the program, you should identify the following key points:

. The program instantiates the required ``llm`` and ``graph`` objects using the appropriate API and connection details.

```
+
[source,python]
----
include::code/cypher-gen.py[tag=openai-neo4j]
----
```

. The ``CYPHER_GENERATION_TEMPLATE`` gives the LLM context. The schema and question are passed to the LLM as input variables.

```
+
[source,python]
----
include::code/cypher-gen.py[tag=template]
```

```
include::code/cypher-gen.py[tag=prompt]
----
```

```
+
The `schema` will be automatically generated from the graph database and passed to the LLM. The `question` will be the user's question.
```

. The program instantiates the ``GraphCypherQChain`` chain with the ``llm``, ``graph``, and prompt template (``cypher_prompt``).

```
+
[source,python]
----
include::code/cypher-gen.py[tag=cypher-chain]
----
```

```
+
```

The program sets the `verbose` flag to `True` so you can see the generated Cypher query and response.

. The chain runs, passing an appropriate question.

+

```
[source,python]
```

```
----
```

```
include::code/cypher-gen.py[tag=invoke]
```

```
----
```

Experiment with different questions and observe the results.

For example, try:

. A different context - "What movies did Meg Ryan act in?"

. An aggregate query - "How many movies has Tom Hanks directed?"

== Inconsistent Results

Investigate what happens when you ask the same question multiple times. Observe the generated Cypher query and the response.

```
"What role did Tom Hanks play in Toy Story?"
```

You will likely see different results each time you run the program.

```
MATCH (actor:Actor {name: 'Tom Hanks'})-[:ACTED_IN]->(movie:Movie
{title: 'Toy Story'})
```

```
RETURN actor.name, movie.title, movie.year, movie.runtime, movie.plot
```

```
MATCH (a:Actor {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie {title: 'Toy
Story'})-[:ACTED_IN]->(p:Person)
```

```
RETURN p.name AS role
```

The LLM doesn't return consistent results - its objective is to produce an answer, not the same response.

The response may not be correct or even generate an error due to invalid Cypher.

In the following two lessons, you will learn how to provide additional context and instructions to the LLM to generate better and more consistent results.

== Check Your Understanding

```
include::questions/1-cypher-chain.adoc[leveloffset=+1]
```

```
[.summary]
```

== Summary

In this lesson, you learned how to use a language model to generate Cypher queries.

In the next lesson, you will experiment with different prompts to improve the results.