Movie Graph Database with Neo4j and Groq LLM

Author: Bhaskar Shivaji Kumbhar

Date: 19 March 2025

Description: This project sets up a graph-based movie database using Neo4j and integrates it with Groq LLM to allow natural language queries on movie data.

Installation and Setup

```
!pip install --upgrade --quiet  langchain langchain-community
langchain-groq neo4j langchain_experimental

## Neo4J Graph DB Information
NEO4J_URI="neo4j+s://XXXXXXXXXXX.neo4j.io"
NEO4J_USERNAME="neo4j"
NEO4J_PASSWORD="XXXXXXXXXXXXXXX"
# Email - qjvdpzrnngcpnbxacq@ytnhy.com

import os
os.environ["NEO4J_URI"]=NEO4J_URI
os.environ["NEO4J_USERNAME"]=NEO4J_USERNAME
os.environ["NEO4J_PASSWORD"]=NEO4J_PASSWORD

from langchain_community.graphs import Neo4jGraph
graph=Neo4jGraph(
    url=NEO4J_URI,
    username=NEO4J_USERNAME,
    password=NEO4J_PASSWORD,
)

<ipython-input-4-18767fd1e585>:2: LangChainDeprecationWarning: The
class `Neo4jGraph` was deprecated in LangChain 0.3.8 and will be
removed in 1.0. An updated version of the class exists in
the :class:`~langchain-neo4j package and should be used instead. To
use it run `pip install -U :class:`~langchain-neo4j` and import as
`from :class:`~langchain_neo4j import Neo4jGraph``.
  graph=Neo4jGraph(

graph

<langchain_community.graphs.neo4j_graph.Neo4jGraph at 0x7bd5a1e91790>
```

Groq Key

```
groq_api_key = "XXXXXXXXXXXXXXXXXX"
```

```
from langchain_groq import ChatGroq

llm = ChatGroq(groq_api_key=groq_api_key, model_name="llama-3.3-70b-
versatile")
llm
```

```
ChatGroq(client=<groq.resources.chat.completions.Completions object at
0x7bd56a8aeb10>,
async_client=<groq.resources.chat.completions.AsyncCompletions object
at 0x7bd56a59ae10>, model_name='llama-3.3-70b-versatile',
model_kwargs={}, groq_api_key=SecretStr('**********'))
```

Loadng the Movie Dataset

```
### Loading the Movie Dataset

movie_query="""
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/tomasonjo/blog-datasets/main/movies
/movies_small.csv' as row

MERGE(m:Movie{id:row.movieId})
SET m.released = date(row.released),
    m.title = row.title,
    m.imdbRating = toFloat(row.imdbRating)
FOREACH (director in split(row.director, '|') |
    MERGE (p:Person {name:trim(director)})
    MERGE (p)-[:DIRECTED]->(m))
FOREACH (actor in split(row.actors, '|') |
    MERGE (p:Person {name:trim(actor)})
    MERGE (p)-[:ACTED_IN]->(m))
FOREACH (genre in split(row.genres, '|') |
    MERGE (g:Genre {name:trim(genre)})
    MERGE (m)-[:IN_GENRE]->(g))
"""

graph
```

```
<langchain_community.graphs.neo4j_graph.Neo4jGraph at 0x7bd5a1e91790>
```

```
graph.query(movie_query)
```

```
[]
```

```
graph.refresh_schema()
print(graph.schema)
```

```
Node properties:
Movie {id: STRING, released: DATE, title: STRING, imdbRating: FLOAT}
Person {name: STRING}
Genre {name: STRING}
```

```
Relationship properties:

The relationships:
(:Movie)-[:IN_GENRE]->(:Genre)
(:Person)-[:DIRECTED]->(:Movie)
(:Person)-[:ACTED_IN]->(:Movie)
```

```
from langchain.chains import GraphCypherQAChain
chain=GraphCypherQAChain.from_llm(llm=llm,graph=graph,verbose=True,all
ow_dangerous_requests=True )
chain
```

```
GraphCypherQAChain(verbose=True,
graph=<langchain_community.graphs.neo4j_graph.Neo4jGraph object at
0x7bd5a1e91790>, cypher_generation_chain=LLMChain(verbose=False,
prompt=PromptTemplate(input_variables=['question', 'schema'],
input_types={}, partial_variables={}, template='Task:Generate Cypher
statement to query a graph database.\nInstructions:\nUse only the
provided relationship types and properties in the schema.\nDo not use
any other relationship types or properties that are not provided.\
nSchema:\n{schema}\nNote: Do not include any explanations or apologies
in your responses.\nDo not respond to any questions that might ask
anything else than for you to construct a Cypher statement.\nDo not
include any text except the generated Cypher statement.\n\nThe
question is:\n{question}'),
llm=ChatGroq(client=<groq.resources.chat.completions.Completions
object at 0x7bd56a8aeb10>,
async_client=<groq.resources.chat.completions.AsyncCompletions object
at 0x7bd56a59ae10>, model_name='llama-3.3-70b-versatile',
model_kwargs={}, groq_api_key=SecretStr('**********')),
output_parser=StrOutputParser(), llm_kwargs={}),
qa_chain=LLMChain(verbose=False,
prompt=PromptTemplate(input_variables=['context', 'question'],
input_types={}, partial_variables={}, template="You are an assistant
that helps to form nice and human understandable answers.\nThe
information part contains the provided information that you must use
to construct an answer.\nThe provided information is authoritative,
you must never doubt it or try to use your internal knowledge to
correct it.\nMake the answer sound as a response to the question. Do
not mention that you based the result on the given information.\nHere
is an example:\n\nQuestion: Which managers own Neo4j stocks?\nContext:
[manager:CTL LLC, manager:JANE STREET GROUP LLC]\nHelpful Answer: CTL
LLC, JANE STREET GROUP LLC owns Neo4j stocks.\n\nFollow this example
when generating answers.\nIf the provided information is empty, say
that you don't know the answer.\nInformation:\n{context}\n\nQuestion:
{question}\nHelpful Answer:"),
llm=ChatGroq(client=<groq.resources.chat.completions.Completions
object at 0x7bd56a8aeb10>,
async_client=<groq.resources.chat.completions.AsyncCompletions object
at 0x7bd56a59ae10>, model_name='llama-3.3-70b-versatile',
model_kwargs={}, groq_api_key=SecretStr('**********')),
output_parser=StrOutputParser(), llm_kwargs={}), graph_schema='Node
properties are the following:\nMovie {id: STRING, released: DATE,
title: STRING, imdbRating: FLOAT},Person {name: STRING},Genre {name:
STRING}\nRelationship properties are the following:\n\nThe
relationships are the following:\n(:Movie)-[:IN_GENRE]->(:Genre),
(:Person)-[:DIRECTED]->(:Movie),(:Person)-[:ACTED_IN]->(:Movie)',
allow_dangerous_requests=True)
```

Responses generated by LLM using Groq API

```
response=chain.invoke({"query":"Who was the director of the moview
GoldenEye"})
response



> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (p:Person)-[:DIRECTED]->(m:Movie {title: "GoldenEye"}) RETURN
p.name
Full Context:
[{'p.name': 'Martin Campbell'}]

> Finished chain.

{'query': 'Who was the director of the moview GoldenEye',
 'result': 'Martin Campbell was the director of the movie GoldenEye.'}

response=chain.invoke({"query":"tell me the genre of th movie
GoldenEye"})
response



> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (m:Movie {title: "GoldenEye"})-[:IN_GENRE]->(g:Genre) RETURN
g.name
Full Context:
[{'g.name': 'Adventure'}, {'g.name': 'Action'}, {'g.name':
'Thriller'}]

> Finished chain.

{'query': 'tell me the genre of th movie GoldenEye',
 'result': 'The genres of the movie GoldenEye are Adventure, Action,
and Thriller.'}

response=chain.invoke({"query":"Who was the director in movie
Casino"})
```

```
response



> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (p:Person)-[:DIRECTED]->(m:Movie {title: 'Casino'}) RETURN
p.name
Full Context:
[{'p.name': 'Martin Scorsese'}]

> Finished chain.

{'query': 'Who was the director in movie Casino',
 'result': 'Martin Scorsese was the director in the movie Casino.'}

response=chain.invoke({"query":"Which movie were released in 2008"})

response



> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (m:Movie) WHERE m.released = "2008" RETURN m.title
Full Context:
[]

> Finished chain.

{'query': 'Which movie were released in 2008',
 'result': "I don't know the answer."}

response=chain.invoke({"query":"Give me the list of movie having imdb
rating more than 8"})
response



> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (m:Movie) WHERE m.imdbRating > 8 RETURN m.title, m.imdbRating
Full Context:
[{'m.title': 'Toy Story', 'm.imdbRating': 8.3}, {'m.title': 'Heat',
'm.imdbRating': 8.2}, {'m.title': 'Casino', 'm.imdbRating': 8.2},
{'m.title': 'Twelve Monkeys (a.k.a. 12 Monkeys)', 'm.imdbRating':
8.1}, {'m.title': 'Seven (a.k.a. Se7en)', 'm.imdbRating': 8.6},
{'m.title': 'Usual Suspects, The', 'm.imdbRating': 8.6}, {'m.title':
'Hate (Haine, La)', 'm.imdbRating': 8.1}, {'m.title': 'Braveheart',
'm.imdbRating': 8.4}, {'m.title': 'Taxi Driver', 'm.imdbRating': 8.3},
{'m.title': 'Anne Frank Remembered', 'm.imdbRating': 8.2}]
```

```
> Finished chain.

{'query': 'Give me the list of movie having imdb rating more than 8',
 'result': 'The movies with an IMDB rating more than 8 are: Seven
(a.k.a. Se7en) with a rating of 8.6, Usual Suspects, The with a rating
of 8.6, Braveheart with a rating of 8.4, Toy Story with a rating of
8.3, and Taxi Driver with a rating of 8.3.'}
```