Neo4j Property Graph Index

Open in Colab

Neo4j is a production-grade graph database, capable of storing a property graph, performing vector search, filtering, and more.

The easiest way to get started is with a cloud-hosted instance using Neo4j Aura

For this notebook, we will instead cover how to run the database locally with docker.

If you already have an existing graph, please skip to the end of this notebook.

```
%pip install llama-index llama-index-graph-stores-neo4j
```

Docker Setup

To launch Neo4j locally, first ensure you have docker installed. Then, you can launch the database with the following docker command

```
docker run \
    -p 7474:7474 -p 7687:7687 \
    -v $PWD/data:/data -v $PWD/plugins:/plugins \
    --name neo4j-apoc \
    -e NEO4J_apoc_export_file_enabled=true \
    -e NEO4J_apoc_import_file_enabled=true \
    -e NEO4J_apoc_import_file_use__neo4j__config=true \
    -e NEO4JLABS_PLUGINS=\[\"apoc\"\] \
    neo4j:latest
```

From here, you can open the db at http://localhost:7474/. On this page, you will be asked to sign in. Use the default username/password of `neo4j` and `neo4j`.

Once you login for the first time, you will be asked to change the password.

After this, you are ready to create your first property graph!

Env Setup

We need just a few environment setups to get started.

```
import os

os.environ["OPENAI_API_KEY"] = "sk-proj-..."
```

```
!mkdir -p 'data/paul_graham/'
!wget 'https://raw.githubusercontent.com/run-
llama/llama_index/main/docs/docs/examples/data/paul_graham/paul_graha
-O 'data/paul_graham/paul_graham_essay.txt'
```

```
import nest_asyncio

nest_asyncio.apply()
```

```
from llama_index.core import SimpleDirectoryReader

documents =
SimpleDirectoryReader("./data/paul_graham/").load_data()
```

```
/Users/loganmarkewich/Library/Caches/pypoetry/virtualenvs/llama-index
-caVs7DDe-py3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWa
rning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Index Construction

```
from llama_index.graph_stores.neo4j import Neo4jPropertyGraphStore

# Note: used to be `Neo4jPGStore`
graph_store = Neo4jPropertyGraphStore(
    username="neo4j",
    password="llamaindex",
```

```
    url="bolt://localhost:7687",
)
```

```
from llama_index.core import PropertyGraphIndex
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.llms.openai import OpenAI
from llama_index.core.indices.property_graph import
SchemaLLMPathExtractor

index = PropertyGraphIndex.from_documents(
    documents,
    embed_model=OpenAIEmbedding(model_name="text-embedding-3-
small"),
    kg_extractors=[
        SchemaLLMPathExtractor(
            llm=OpenAI(model="gpt-3.5-turbo", temperature=0.0)
        )
    ],
    property_graph_store=graph_store,
    show_progress=True,
)
```

```
Parsing nodes: 100%|██████████| 1/1 [00:00<00:00, 21.63it/s]
Extracting paths from text with schema: 100%|██████████| 22/22 [01:06<
00:00,  3.02s/it]
Generating embeddings: 100%|██████████| 1/1 [00:00<00:00,  1.06it/s]
Generating embeddings: 100%|██████████| 1/1 [00:00<00:00,  1.89it/s]
```

Now that the graph is created, we can explore it in the UI by visting http://localhost:7474/.

The easiest way to see the entire graph is to use a cypher command like `"match n=() return n"` at the top.

To delete an entire graph, a useful command is `"match n=() detach delete n"`.

Querying and Retrieval

```
retriever = index.as_retriever(
    include_text=False,  # include source text in returned nodes,
default True
)

nodes = retriever.retrieve("What happened at Interleaf and
```

```
Viaweb?")

for node in nodes:
    print(node.text)
```

```
Interleaf -> Got crushed by -> Moore's law
Interleaf -> Made -> Scripting language
Interleaf -> Had -> Smart people
Interleaf -> Inspired by -> Emacs
Interleaf -> Had -> Few years to live
Interleaf -> Made -> Software
Interleaf -> Had done -> Something bold
Interleaf -> Added -> Scripting language
Interleaf -> Built -> Impressive technology
Interleaf -> Was -> Company
Viaweb -> Was -> Profitable
Viaweb -> Was -> Growing rapidly
Viaweb -> Suggested -> Hospital
Idea -> Was clear from -> Experience
Idea -> Would have to be embodied as -> Company
Painting department -> Seemed to be -> Rigorous
```

```
query_engine = index.as_query_engine(include_text=True)

response = query_engine.query("What happened at Interleaf and
Viaweb?")

print(str(response))
```

```
Interleaf had smart people and built impressive technology but got cr
ushed by Moore's Law. Viaweb was profitable and growing rapidly.
```

Loading from an existing Graph

If you have an existing graph (either created with LlamaIndex or otherwise), we can connect to and use it!

NOTE: If your graph was created outside of LlamaIndex, the most useful retrievers will be text to cypher or cypher templates. Other retrievers rely on properties that LlamaIndex inserts.

```
from llama_index.graph_stores.neo4j import Neo4jPropertyGraphStore
from llama_index.core import PropertyGraphIndex
from llama_index.embeddings.openai import OpenAIEmbedding
```

```
from llama_index.llms.openai import OpenAI

graph_store = Neo4jPropertyGraphStore(
    username="neo4j",
    password="794613852",
    url="bolt://localhost:7687",
)

index = PropertyGraphIndex.from_existing(
    property_graph_store=graph_store,
    llm=OpenAI(model="gpt-3.5-turbo", temperature=0.3),
    embed_model=OpenAIEmbedding(model_name="text-embedding-3-
small"),
)
```

From here, we can still insert more documents!

```
from llama_index.core import Document

document = Document(text="LlamaIndex is great!")

index.insert(document)
```

```
nodes =
index.as_retriever(include_text=False).retrieve("LlamaIndex")

print(nodes[0].text)
```

```
Llamaindex -> Is -> Great
```

For full details on construction, retrieval, querying of a property graph, see the full docs page.