

```
= Retrievers
:order: 8
:type: lesson
:disable-cache: true
```

link:<https://python.langchain.com/v0.2/docs/integrations/retrievers/>[Retrievers^] are Langchain chain components that allow you to retrieve documents using an unstructured query.

Find a movie plot about a robot that wants to be human.

Documents are any unstructured text that you want to retrieve. A retriever often uses a vector store as its underlying data structure.

Retrievers are a key component for creating models that can take advantage of Retrieval Augmented Generation (RAG).

Previously, you loaded embeddings and created a vector index of Movie plots - in this example, the movie plots are the `_documents_`, and a `_retriever_` could be used to give a model context.

In this lesson, you will create a `_retriever_` to retrieve documents from the movie plots vector index.

```
== Neo4jVector
```

The link:<https://python.langchain.com/v0.2/docs/integrations/vectorstores/neo4jvector/>[`Neo4jVector`] is a Langchain vector store that uses a Neo4j database as the underlying data structure.

You can use the ``Neo4jVector`` to generate embeddings, store them in the database and retrieve them.

```
=== Querying a vector index
```

Review the following code that creates a ``Neo4jVector`` from the ``moviePlots`` index you created.

```
[source,python]
----
include::code/query_vector.py[]
----
```

You should be able to identify the following:

- \* That an ``embedding_provider`` is required. In this case, ``OpenAIEmbeddings`` created the embeddings for the movie plots. The embedding provider will also generate embeddings for any queries.
- \* The connection to the Neo4j database (``graph``).
- \* The name of the Neo4j index (``"moviePlots"``).
- \* The name of the node property that contains the embeddings (``"plotEmbedding"``).

- \* The name of the node property that contains the text (`"plot"`).
- \* The `similarity\_search()` method is used to retrieve documents. The first argument is the query.

To run this program, you will need to:

- . Replace the `openai\_api\_key` with your OpenAI API key
- . Update Neo4j connection details with your Sandbox 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}#
=====
```

Run the code and review the results. Try different queries and see what results you get.

The `similarity\_search()` method returns a list of `link:https://api.python.langchain.com/en/latest/documents/langchain_core.documents.base.Document.html[Document^]` objects. The `Document` object includes the properties:

- \* `page\_content` - the content referenced by the index, in this example the plot of the movie
- \* `meta\_data` - a dictionary of the `Movie` node properties returned by the index

[TIP]  
.Specify the number of documents  
=====

You can pass an optional `k` argument to the `similarity\_search()` method to specify the number of documents to return. The default is 4.

```
[source,python]
----
vector.similarity_search(query, k=1)
----
=====
```

=== Creating a new vector index

The `Neo4jVector` class can also generate embeddings and vector indexes - this is useful when creating vectors programmatically or at run time.

The following code would create embeddings and a new index called `myVectorIndex` in the database for `Chunk` nodes with a `text` property:

```
[source,python]
----
include::code/create_index.py[ ]
```

----

If you would like to know more about creating vectors for unstructured data and documents in Neo4j, check out the GraphAcademy course [link:https://graphacademy.neo4j.com/courses/llm-vectors-unstructured/](https://graphacademy.neo4j.com/courses/llm-vectors-unstructured/)[Introduction to Vector Indexes and Unstructured Data^].

== Creating a Retriever chain

To incorporate a retriever and Neo4j vector into a Langchain application, you can create a `_retrieval_` chain.

The ``Neo4jVector`` class has a ``as_retriever()`` method that returns a retriever.

The [link:https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval\\_qa.base.RetrievalQA.html#`RetrievalQA`](https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval_qa.base.RetrievalQA.html#`RetrievalQA`) class is a chain that uses a retriever as part of its pipeline. It will use the retriever to retrieve documents and pass them to a language model.

By incorporating ``Neo4jVector`` into a ``RetrievalQA`` chain, you can use data and vectors in Neo4j in a Langchain application.

Review this program incorporating the ``moviePlots`` vector index into a retrieval chain.

[source,python]

----

```
include::code/retriever_chain.py[]
```

----

When the program runs, the ``RetrievalQA`` chain will use the ``movie_plot_vector`` retriever to retrieve documents from the ``moviePlots`` index and pass them to the ``chat_llm`` language model.

[TIP]

.Understanding the results

====

It can be difficult to understand how the model generated the response and how the retriever affected it.

By setting the optional ``verbose`` and ``return_source_documents`` arguments to ``True`` when creating the ``RetrievalQA`` chain, you can see the source documents and the retriever's score for each document.

[source, python]

----

```
plot_retriever = RetrievalQA.from_llm(  
    llm=chat_llm,  
    retriever=movie_plot_vector.as_retriever(),  
    verbose=True,  
    return_source_documents=True
```

```
)  
----  
====
```

Retrievers and vector indexes allow you to incorporate unstructured data into your Langchain applications.

== Check Your Understanding

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

```
[.summary]  
== Summary
```

In this lesson, you learned how to incorporate Neo4j vector indexes and retrievers into Langchain applications.

In the next optional challenge, you will add the movie plots vector retriever to the chat agent you created in the previous lesson.