

```
= Storing Conversation History
:order: 6
:type: lesson
:disable-cache: true
```

Langchain includes functionality to integrate directly with Neo4j, including allowing you to run Cypher statements, query vector indexes and use Neo4j as a conversation memory store.

In this lesson, you will learn how to connect to and use a Neo4j database as a conversation memory store.

Storing conversation history in a Neo4j database allows you to analyze the conversation history to understand trends and improve outcomes.

== Connecting to a Neo4j instance

The following code will connect to a Neo4j database and run a simple query.

```
[source,python]
----
include::code/connect-to-neo4j.py[]
----
```

You can connect to the Neo4j sandbox created for you when you joined the course.

Update the code above to use the `url`, `username` and `password` of your Neo4j sandbox.

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

Run the query - you should see data about the movie Toy Story.

The `Neo4jGraph` class is a wrapper to the [link:https://neo4j.com/docs/python-manual/current/\[Neo4j Python driver^\]](https://neo4j.com/docs/python-manual/current/[Neo4j Python driver^]). It simplifies connecting to Neo4j and integrating with the Langchain framework.

=== Schema

When you connect to the Neo4j database, the object loads the database schema into memory - this enables Langchain to access the schema information without having to query the database.

You can access the schema information using the `schema` property.

```
[source,python]
----
```

```
print(graph.schema)
----
```

[TIP]

.Refreshing the schema

You can refresh the schema by calling the `graph.refresh_schema()` method.

== Conversation History

In the previous lesson, you created a program that used the `ChatMessageHistory` component to store conversation history in memory.

You will now update this program to store the conversation history in your Neo4j sandbox using the `Neo4jChatMessageHistory` component.

[%collapsible]

.Reveal the code

====

[source,python]

----

```
include:../3.5-memory/code/chat-model-memory.py[tag=**]
```

----

====

=== Session ID

You must create and assign a session ID to each conversation to identify them.

The session ID can be any unique value, such as a link:[https://en.wikipedia.org/wiki/Universally\\_unique\\_identifier](https://en.wikipedia.org/wiki/Universally_unique_identifier)[Universally Unique Identifier (UUID)^].

You can generate a random UUID using the Python `uuid.uuid4` function.

Create a new `SESSION_ID` constant in your chat model program.

[source,python]

----

```
include:code/chat-model-memory-neo4j.py[tag=session-id]
```

----

This session ID will be used to identify the conversation in Neo4j.

=== Neo4j Chat Message History

Create a `Neo4jGraph` object to connect to your Neo4j sandbox.

[source,python]

----

```
include:code/chat-model-memory-neo4j.py[tag=import-neo4j]
```

```
include:code/chat-model-memory-neo4j.py[tag=neo4j-graph]
```

----

Remember to update the connection details with your Neo4j sandbox 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}#

====

Previously, the ``get_memory`` function returned an instance of ``ChatMessageHistory``.

The ``get_memory`` function should now return an instance of ``Neo4jChatMessageHistory``.

You should pass the ``session_id`` and the ``graph`` connection you created as parameters.

[source,python]

----

include::code/chat-model-memory-neo4j.py[tag=import-neo4j-chat]

include::code/chat-model-memory-neo4j.py[tag=get-memory]

----

Finally, you must add the ``SESSION_ID`` to the ``config`` when you ``invoke`` the chat model.

[source,python]

----

include::code/chat-model-memory-neo4j.py[tag=invoke]

----

[%collapsible]

.Click to reveal the complete code.

====

[source,python]

----

include::code/chat-model-memory-neo4j.py[tag=\*\*]

----

====

Run the program and have a conversation with the chat model.  
The conversation history will now be stored in your Neo4j sandbox.

=== Conversation History Graph

The conversation history is stored using the following data model:

image::images/Neo4jChatMessageHistory.png[A graph data model showing 2 nodes Session and Message connected by a LAST\_MESSAGE relationship. There

is a circular NEXT relationship on the Message node.]

The `Session` node represents a conversation session and has an `id` property.

The `Message` node represents a message in the conversation and has the following properties:

- \* `content` - The message content
- \* `type` - The message type: `human`, `ai`, or `system`

The `LAST\_MESSAGE` relationship connects the `Session` node to the conversation's last `Message` node. The `NEXT` relationship connects `Message` nodes in the conversation.

You can return the graph of the conversation history using the following Cypher query:

```
[source, cypher]
----
MATCH (s:Session)-[:LAST_MESSAGE]->(last:Message)<-[:NEXT*]-(msg:Message)
RETURN s, last, msg
----
```

image::images/conversation-graph.svg[A graph showing a Session node connected to a Message through with a LAST\_MESSAGE relationship. Message nodes are connected to each other with NEXT relationships.]

You can return the conversation history for a single session by filtering on the `Session.id` property.

```
[source, cypher]
----
MATCH (s:Session)-[:LAST_MESSAGE]->(last:Message)
WHERE s.id = 'your session id'
MATCH p = (last)<-[:NEXT*]-(msg:Message)
UNWIND nodes(p) as msgs
RETURN DISTINCT msgs.type, msgs.content
----
```

== Check Your Understanding

include::questions/1-neo4jgraph.adoc[leveloffset=+1]

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

In this lesson, you learned how to use a Neo4j database as a conversation memory store.

In the next lesson, you will learn how to create an agent to give an LLM access to different tools and data sources.