```
= Vector Indexes
:order: 3
:type: lesson
:sandbox: true

[.slide.discrete]
== Vector Indexes

In the last lesson, you learned about embeddings, vectors and their role
in RAG.

In this lesson, you will learn how to use a vector index in Neo4j to
compare embeddings to find similar data.

[.slide]
== Movie Plots

GraphAcademy created a Neo4j sandbox of movie recommendations when you
enrolled in this course.
The recommendations database contains over 9000 movies, 15000 actors, and
over 100000 user ratings.

Each movie has a `.plot` property.

.Movie Plot Example
[source,cypher]
MATCH (m:Movie {title: "Toy Story"})
RETURN m.title AS title, m.plot AS plot

    "A cowboy doll is profoundly threatened and jealous when a new
spaceman figure supplants him as top toy in a boy's room."

[.slide]
== Plot Embeddings

Embeddings have been created for 1000 movie plots.
The embedding is stored in the `.plotEmbedding` property of the `Movie`
nodes.

.View the plot embedding
[source,cypher]
MATCH (m:Movie {title: "Toy Story"})
RETURN m.title AS title, m.plot AS plot, m.plotEmbedding

[.transcript-only]
====
The following Cypher query will return the titles and plots for the movies
that have embeddings:

[source, cypher]
----
MATCH (m:Movie)
WHERE m.plotEmbedding IS NOT NULL
```

```
RETURN m.title, m.plot
----
====

A vector index, `moviePlots`, has been created for the `.plotEmbedding`
property of the `Movie` nodes.

You can use the `moviePlots` vector index to find the most similar movies
by comparing the movie plot embeddings.

[.transcript-only]
====

[%collapsible]
.Click to see how the vector index was created
=====
This Cypher script loads the Movie plot embeddings from an external file
and create the `moviePlots` vector index:

[source, cypher]
----
include::./reset.cypher[]
----
=====

[TIP]
=====
You can learn more about creating embeddings and vector indexes in the
link:https://graphacademy.neo4j.com/courses/llm-vectors-unstructured[Graph
Academy Introduction to Vector Indexes and Unstructured Data course^].
=====
====

[.slide.col-2]
== Querying Vector Indexes

[.col]
====
You can query the `moviePlots` index using the
link:https://neo4j.com/docs/cypher-manual/current/indexes/semantic-indexes
/vector-indexes/#query-vector-index[`db.index.vector.queryNodes()`^]
procedure.

The procedure returns the requested number of approximate nearest neighbor
nodes and their similarity score, ordered by the score.

.db.index.vector.queryNodes Syntax
[source,cypher,role=nocopy noplay]
----
CALL db.index.vector.queryNodes(
    indexName :: STRING,
    numberOfNearestNeighbours :: INTEGER,
    query :: LIST<FLOAT>
```

```
) YIELD node, score
----
====

[.col]
====
The procedure accepts three parameters:

. `indexName` - The name of the vector index
. `numberOfNearestNeighbours` - The number of results to return
. `query` - A list of floats that represent an embedding

The procedure yields two arguments:

. A `node` which matches the query
. A similarity `score` ranging from `0.0` to `1.0`.

You can use this procedure to find the closest embedding value to a given
value.
====

[.slide.col-60-40]
== Querying Similar Movie Plots

[.col]
====
You can use the `moviePlots` vector index to find movies with similar
plots.

Review this Cypher before running it.

[source,cypher]
.Similar Plots
----
MATCH (m:Movie {title: 'Toy Story'})

CALL db.index.vector.queryNodes('moviePlots', 6, m.plotEmbedding)
YIELD node, score

RETURN node.title AS title, node.plot AS plot, score
----
====

[.col]
====
The query finds the _Toy Story_ `Movie` node and uses the `.plotEmbedding`
property to find the most similar plots.

The `db.index.vector.queryNodes()` procedure uses the `moviePlots` vector
index to find similar embeddings.
====

[.transcript-only]
```

```
====
Run the query. The procedure returns the requested number of nodes and
their similarity score, ordered by the score.

[%collapsible]
.Click to reveal the results
=====
.Similar Plots Results
|===
| title | plot | score
| "Toy Story" | "A cowboy doll is profoundly threatened and jealous when a
new spaceman figure supplants him as top toy in a boy's room." | 1.0
| "Little Rascals, The" | "Alfalfa is wooing Darla and his
He-Man-Woman-Hating friends attempt to sabotage the relationship." |
0.9214372634887695
| "NeverEnding Story III, The" | "A young boy must restore order when a
group of bullies steal the magical book that acts as a portal between
Earth and the imaginary world of Fantasia." | 0.9206198453903198
|  "Drop Dead Fred" | "A young woman finds her already unstable life
rocked by the presence of a rambunctious imaginary friend from childhood."
| 0.9199690818786621
| "E.T. the Extra-Terrestrial" | "A troubled child summons the courage to
help a friendly alien escape Earth and return to his home-world." |
0.919100284576416
| "Gumby: The Movie" | "In this offshoot of the 1950s claymation cartoon
series, the crazy Blockheads threaten to ruin Gumby's benefit concert by
replacing the entire city of Clokeytown with robots." | 0.9180967211723328
|===
=====

The similarity score is between `0.0` and `1.0`, with `1.0` being the most
similar. Note how the most similar plot is that of the _Toy Story_ movie
itself!
====

[.slide.col-60-40]
== Generate Embeddings

[.col]
====
You can generate a new embedding in Cypher using the
link:https://neo4j.com/docs/cypher-manual/current/genai-integrations/#sing
le-embedding[`genai.vector.encode`^] function:

.genai.vector.encode Syntax
[source, cypher]
----
WITH genai.vector.encode(
    "Text to create embeddings for",
    "OpenAI",
    { token: "sk-..." }) AS embedding
RETURN embedding
----
```

```
====

[.col]
====
[IMPORTANT]
=====
You will need to replace `token: "sk-..."` with an
link:https://platform.openai.com/api-keys[OpenAI API key^].
=====
====

[.slide]
== Generate a Plot Embedding

You can use the embedding to query the vector index to find similar
movies.

This query, creates and embedding for the text _"A mysterious spaceship
lands Earth"_ and uses it to query the `moviePlots` vector index for the 6
most similar movie plots.

[source, cypher]
----
WITH genai.vector.encode(
    "A mysterious spaceship lands Earth",
    "OpenAI",
    { token: "sk-..." }) AS myMoviePlot
CALL db.index.vector.queryNodes('moviePlots', 6, myMoviePlot)
YIELD node, score
RETURN node.title, node.plot, score
----

Experiment with different movie plots and observe the results.

[.slide]
== Considerations

Using embeddings and vectors is relatively straightforward and can quickly
yield results.
The downside to this approach is that it relies heavily on the embeddings
and similarity function to produce valid results.

This approach is also a black box.
There are 1536 dimensions; it would be impossible to determine how the
vectors are structured and how they influenced the similarity score.

The movies returned look similar, but without reading and comparing them,
you would have no way of verifying that the results are correct.

[.slide.discrete]
== Considerations

Vectors work well for:
```

* Contextual or Meaning Based Questions
* Fuzzy or Vague queries
* Broad or Open-Ended questions
* Complex queries with multiple concepts

Vectors are ineffective for:

* Highly Specific or Fact-Based Questions
* Numerical or Exact-Match Queries
* Boolean or Logical Queries
* Ambiguous or Unclear Queries without Context
* Specialised Knowledge

In the next lesson you will look at how you can improve the results by using a combination of vector and graph queries.

[.next]
== Check your understanding

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

[.summary]
== Lesson Summary

In this lesson, you learned how to use a vector index in Neo4j and when they are useful for finding context for Generative AI applications.

In the next lesson, you will learn how to GraphRAG can improve the results of your queries.