Lesson 3: Preparing Text Data for RAG

Note: This notebook takes about 30 seconds to be ready to use. Please wait until the "Kernel starting, please wait..." message clears from the top of the notebook before running any cells. You may start the video while you wait.

Import packages and set up Neo4j

In [1]:
```
from dotenv import load_dotenv
import os

from langchain_community.graphs import Neo4jGraph

# Warning control
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Load from environment
load_dotenv('.env', override=True)
NEO4J_URI = os.getenv('NEO4J_URI')
NEO4J_USERNAME = os.getenv('NEO4J_USERNAME')
NEO4J_PASSWORD = os.getenv('NEO4J_PASSWORD')
NEO4J_DATABASE = os.getenv('NEO4J_DATABASE')
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

# Note the code below is unique to this course environment, and not
# standard part of Neo4j's integration with OpenAI. Remove if runnir
# in your own environment.
OPENAI_ENDPOINT = os.getenv('OPENAI_BASE_URL') + '/embeddings'
```

In [3]:
```
# Connect to the knowledge graph instance using LangChain
kg = Neo4jGraph(
    url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD,
)
```

Create a vector index

In [4]:
```
kg.query("""
  CREATE VECTOR INDEX movie_tagline_embeddings IF NOT EXISTS
  FOR (m:Movie) ON (m.taglineEmbedding)
  OPTIONS { indexConfig: {
    `vector.dimensions`: 1536,
    `vector.similarity_function`: 'cosine'
  }}""")
```

[]

In [5]:
```
kg.query("""
  SHOW VECTOR INDEXES
  """
)
```

```
[{'id': 3,
  'name': 'movie_tagline_embeddings',
  'state': 'ONLINE',
  'populationPercent': 100.0,
  'type': 'VECTOR',
  'entityType': 'NODE',
  'labelsOrTypes': ['Movie'],
  'properties': ['taglineEmbedding'],
  'indexProvider': 'vector-1.0',
  'owningConstraint': None,
  'lastRead': neo4j.time.DateTime(2024, 3, 18, 4, 44, 33, 865000000, tzin
fo=<UTC>),
  'readCount': 15}]
```

Populate the vector index

- Calculate vector representation for each movie tagline using OpenAI
- Add vector to the `Movie` node as `taglineEmbedding` property

In [6]:
```
kg.query("""
    MATCH (movie:Movie) WHERE movie.tagline IS NOT NULL
    WITH movie, genai.vector.encode(
        movie.tagline,
        "OpenAI",
        {
          token: $openAiApiKey,
          endpoint: $openAiEndpoint
        }) AS vector
    CALL db.create.setNodeVectorProperty(movie, "taglineEmbedding",
    """,
    params={"openAiApiKey":OPENAI_API_KEY, "openAiEndpoint": OPENAI_
```

```
[]
```

In [7]:
```
result = kg.query("""
    MATCH (m:Movie)
    WHERE m.tagline IS NOT NULL
    RETURN m.tagline, m.taglineEmbedding
    LIMIT 1
    """
)
```

In [8]:
```
result[0]['m.tagline']
```

```
'Welcome to the Real World'
```

In [9]:
```
result[0]['m.taglineEmbedding'][:10]
```

```
[0.01738535612821579,
 -0.005492697935551405,
 -0.002040519379079342,
 -0.02559983730316162,
 -0.01443757489323616,
 0.01673029363155365,
 -0.017123330384492874,
 0.0005064451252110302,
 -0.02524610422551632,
 -0.02953021228313446]
```

In [10]:
```
len(result[0]['m.taglineEmbedding'])
```

```
1536
```

Similarity search

- Calculate embedding for question
- Identify matching movies based on similarity of question and `taglineEmbedding` vectors

In [11]:
```
question = "What movies are about love?"
```

In [12]:
```
result1 = kg.query("""
    WITH genai.vector.encode(
        $question,
        "OpenAI",
        {
          token: $openAiApiKey,
          endpoint: $openAiEndpoint
        }) AS question_embedding
    CALL db.index.vector.queryNodes(
        'movie_tagline_embeddings',
        $top_k,
        question_embedding
        ) YIELD node AS movie, score
    RETURN movie.title, movie.tagline, score
    """,
    params={"openAiApiKey":OPENAI_API_KEY,
            "openAiEndpoint": OPENAI_ENDPOINT,
            "question": question,
            "top_k": 5
            })
```

In [14]:
```
result1
```

```
[{'movie.title': 'Joe Versus the Volcano',
  'movie.tagline': 'A story of love, lava and burning desire.',
  'score': 0.9063377380371094},
 {'movie.title': 'As Good as It Gets',
  'movie.tagline': 'A comedy from the heart that goes for the throat.',
  'score': 0.9022750854492188},
 {'movie.title': 'Snow Falling on Cedars',
  'movie.tagline': 'First loves last. Forever.',
  'score': 0.9013444781303406},
 {'movie.title': 'Sleepless in Seattle',
  'movie.tagline': 'What if someone you never met, someone you never saw,
someone you never knew was the only someone for you?',
  'score': 0.8944939374923706},
 {'movie.title': 'When Harry Met Sally',
  'movie.tagline': 'Can two friends sleep together and still love each ot
her in the morning?',
  'score': 0.8942663669586182}]
```

Option-1: Pythonic

In [15]:
```
len(result1) ## Pythonic
```

```
5
```

Option-2: Through LLM via prompt engineering

In [ ]:
```
# You can also get the count through LLM on result1 as input text.
```

Option-3: Cypher Query by returning count

In [17]:
```
result2 = kg.query("""
    WITH genai.vector.encode(
        $question,
        "OpenAI",
        {
          token: $openAiApiKey,
          endpoint: $openAiEndpoint
        }) AS question_embedding
    CALL db.index.vector.queryNodes(
        'movie_tagline_embeddings',
        $top_k,
        question_embedding
        ) YIELD node AS movie, score
    RETURN count(movie) as movie_count
    """,
    params={"openAiApiKey":OPENAI_API_KEY,
            "openAiEndpoint": OPENAI_ENDPOINT,
            "question": question,
            "top_k": 5
            })
```

In [18]:
```
result2
```

```
[{'movie_count': 5}]
```

In [19]:
```
result2[0].get('movie_count')
```

```
5
```