Phase II – Data Processing

I decided that the best place to start with collecting data for a tool like the one I am building would be the Annual Baseline datasets from PubMed. This gives a large set of articles submitted to PubMed and they are updated yearly.

The data comes in an XML format, and requires processing. The data contains the citations, authors, abstracts, and more for each of the papers in the Annual Baseline. As I collect that data from the XML file, I also use the nltk package for Python to pull keywords from the abstracts of the papers.

I decided the best way to store this data would be using a Neo4J graph database. My code creates nodes from the authors, paper titles, and their extracted keywords allowing me to have a way to link like papers based on their content as well as the teams that put them together. Additionally, since I have extracted keywords, I can even expand the data to connect those keywords to ontology codes, diagnoses, vectorized graphics, and more.

This implementation is sort of a first-shot and will likely continue to be revised as development of the project goes on, but as a starting point before seeing how well it works with the LLM and building out how the LLM can respond based on this data, this is the bulk of what I expected to include.

Sources:

<https://pubmed.ncbi.nlm.nih.gov/download/#annual-baseline>

<https://www.nltk.org/>

<https://neo4j.com/>

Code:

from neo4j import GraphDatabase  
from lxml import etree  
import nltk  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
import re  
  
# Download necessary NLTK data  
nltk.download('punkt')  
nltk.download('stopwords')  
  
# Connect to Neo4j  
uri = "bolt://localhost:7687"  
driver = GraphDatabase.driver(uri, auth=("neo4j", "password"))  
  
  
def create\_article(tx, title, pmid, abstract):  
 tx.run("MERGE (a:Article {pmid: $pmid}) SET a.title = $title, a.abstract = $abstract",  
 title=title, pmid=pmid, abstract=abstract)  
  
  
def create\_author(tx, first\_name, last\_name, pmid):  
 tx.run("""  
 MATCH (a:Article {pmid: $pmid})  
 MERGE (auth:Author {first\_name: $first\_name, last\_name: $last\_name})  
 MERGE (auth)-[:AUTHORED]->(a)  
 """, first\_name=first\_name, last\_name=last\_name, pmid=pmid)  
  
  
def create\_keyword(tx, keyword, pmid):  
 tx.run("""  
 MATCH (a:Article {pmid: $pmid})  
 MERGE (k:Keyword {name: $keyword})  
 MERGE (a)-[:HAS\_KEYWORD]->(k)  
 """, keyword=keyword, pmid=pmid)  
  
  
def extract\_keywords(abstract, n=5):  
 words = word\_tokenize(abstract.lower()) # tokenize, make tokens lowercase  
  
 # Remove stopwords and non-alphabetic tokens  
 stop\_words = set(stopwords.words('english'))  
 words = [word for word in words if word.isalpha() and word not in stop\_words]  
  
 freq\_dist = nltk.FreqDist(words) # frequency distribution  
  
 return [word for word, \_ in freq\_dist.most\_common(n)] # Return the n most common words as keywords  
  
  
def process\_xml\_file(file\_path):  
 # Parse the XML file  
 with open(file\_path, 'rb') as f:  
 tree = etree.parse(f)  
  
 root = tree.getroot()  
  
 # Iterate through PubMed articles contained in the Annual Baseline  
 for article in root.xpath('//PubmedArticle'):  
 title = article.xpath('.//ArticleTitle')[0].text  
 pmid = article.xpath('.//PMID')[0].text  
 abstract = ' '.join(article.xpath('.//AbstractText/text()'))  
  
 print(f"Processing Article - Title: {title}, PMID: {pmid}")  
  
 # Create article node from pubmed baseline  
 with driver.session() as session:  
 session.execute\_write(create\_article, title, pmid, abstract)  
  
 # Process authors from pubmed baseline  
 authors = article.xpath('.//AuthorList/Author')  
 for author in authors:  
 last\_name = author.xpath('.//LastName')  
 first\_name = author.xpath('.//ForeName')  
  
 if last\_name and first\_name:  
 last\_name = last\_name[0].text  
 first\_name = first\_name[0].text  
 print(f"Processing Author: {first\_name} {last\_name}, Article PMID: {pmid}")  
  
 # Create author node and assign relationship  
 with driver.session() as session:  
 session.execute\_write(create\_author, first\_name, last\_name, pmid)  
  
 # Extract and process keywords from paper abstracts  
 if abstract:  
 keywords = extract\_keywords(abstract)  
 for keyword in keywords:  
 print(f"Processing Keyword: {keyword}, Article PMID: {pmid}")  
  
 # Create keyword node and assign relationship(s)  
 with driver.session() as session:  
 session.execute\_write(create\_keyword, keyword, pmid)  
  
  
def main():  
 file\_path = 'pubmed24n0001.xml' # Update this to your file path  
 process\_xml\_file(file\_path)  
 print("Database population complete.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 main()  
 finally:  
 driver.close()