Rationale

MongoDB is the best option for the first query because:

-Using findOne() we can obtain a Single-Document Query that can be executed in one read... to obtain disease names, drug names that can treat or palliate this disease, gene names that cause this disease, and where this disease occurs.

-Denormalized data allows for this because information about the disease is embeded in one document with nested arrays or references to everything connected to that particular disease.

-Since MongoDB was made with denormalized data for a single-document query, the reads from a hierarchical document store are going to be very fast.

Neo4j is the best option for the second query because:

-Using relationships between nodes to pattern match, allows one to find compounds that treat a disease (excluding compounds that treat disease) much faster than a document store would because edges and nodes are clearly defined by relationships.

-One can traverse through these clearly defined relationships quickly due to efficient pattern matching

Aggregation Method for MongoDB

In my project, I used a Python script to import data from two TSV files, nodes.tsv and edges.tsv, into my MongoDB database. First, I connected to MongoDB using the pymongo library and specified my MongoDB URI. Then, I connected to my database named "493". I opened the nodes.tsv file and read its contents using csv.DictReader, which converts each row into a dictionary. I stored all the dictionaries in a list and inserted them into the nodes collection in my database using insert\_many. I repeated the same steps for the edges.tsv file, inserting its data into the edges collection. This way, I successfully imported all the data into MongoDB.

I aggregated data from the two files by using the relationships defined in the edges collection. First, I found the disease name from the nodes collection using the disease ID. Then, I looked for drug information by checking the edges that connect the disease to compounds using the CtD and CpD metaedges, which show which drugs treat or palliate the disease. After that, I gathered gene names related to the disease by looking for edges that use the DdG, DuG, or DaG metaedges. Finally, I found anatomy parts affected by the disease by checking edges with the DlA metaedge. By using these connections between the two collections, I was able to collect all the relevant information into one result.



Potential Improvements for MongoDB

To make my HetioNet project faster and more efficient with MongoDB, I can do a few things. First, I can create indexes on fields that I often search or sort by. This helps the database find what I need quickly without looking through everything.

For large amounts of data, I can use sharding to spread the data across multiple servers. This balances the load and makes reading and writing data faster.

I can also use the aggregation framework to handle complex queries better. It processes data on the server side, which reduces the amount of data that needs to be transferred and speeds up queries. By doing these things, my project will run smoother and be easier to manage.