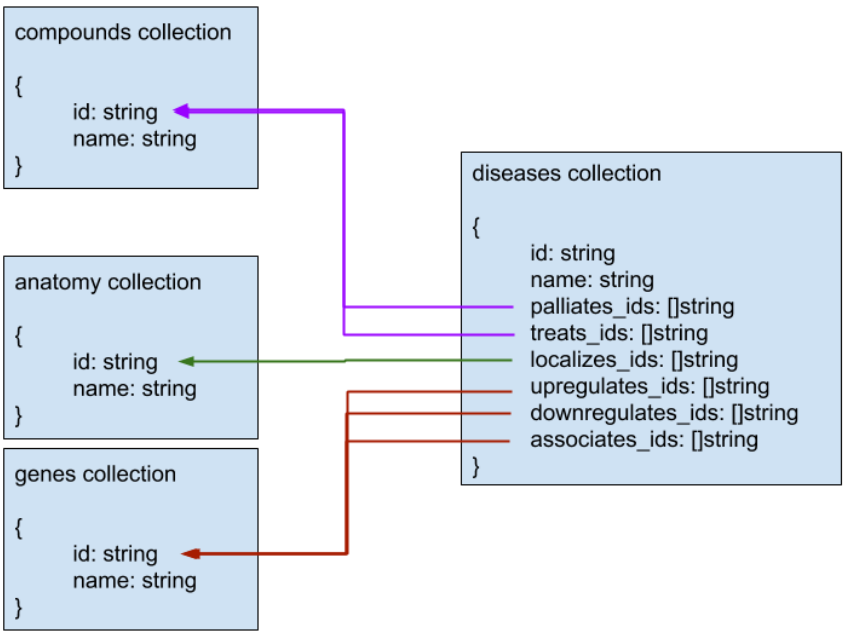
David Margolin
Project 1

This project seeks to create database infrastructure to model HetioNet. Two different NOSQL databases are used to answer different questions.

1. Given a disease, what is its name, what are drug names that can treat or palliate this disease, what are gene names that cause this disease, and where this disease occurs? Obtain and output this information in a single query.

Database Used: MongoDB
- MongoDB is used because the query concerns itself with a single type of item that may contain a really large amount of information (a document)

Design Diagram:



Query:

```
hetio_db["diseases"].aggregate([
        {"$match":{"id":disease_id}},
        {"$limit":1},
        {"$lookup":{"from":"anatomy","localField":"localizes_ids","foreignField":"id","as":"localizes"}},
        {"$lookup":{"from":"compounds","localField":"palliates_ids","foreignField":"id","as":"palliates"}},
        {"$lookup":{"from":"compounds","localField":"treats_ids","foreignField":"id","as":"treats"}},
        {"$lookup":{"from":"genes","localField":"upregulates_ids","foreignField":"id","as":"upregulates"}},
        {"$lookup":{"from":"genes","localField":"downregulates_ids","foreignField":"id","as":"downregulates"}},
        {"$lookup":{"from":"genes","localField":"associates_ids","foreignField":"id","as":"associates"}},
        {"$project":{
                "_id":0,
                "name":1,
                "localizes.name":1,
```

```
                "associates.name":1,
                "treats.name":1,
                "upregulates.name":1,
                "downregulates.name":1,
                "palliates.name":1
        }}
]))
```

Import Nodes:

```
hetio_db[collection].insert_many([
        {"id": $id, "name": $name},
        {"id": $id, "name": $name},
        {"id": $id, "name": $name},
        ...
])
```

Import Edges:

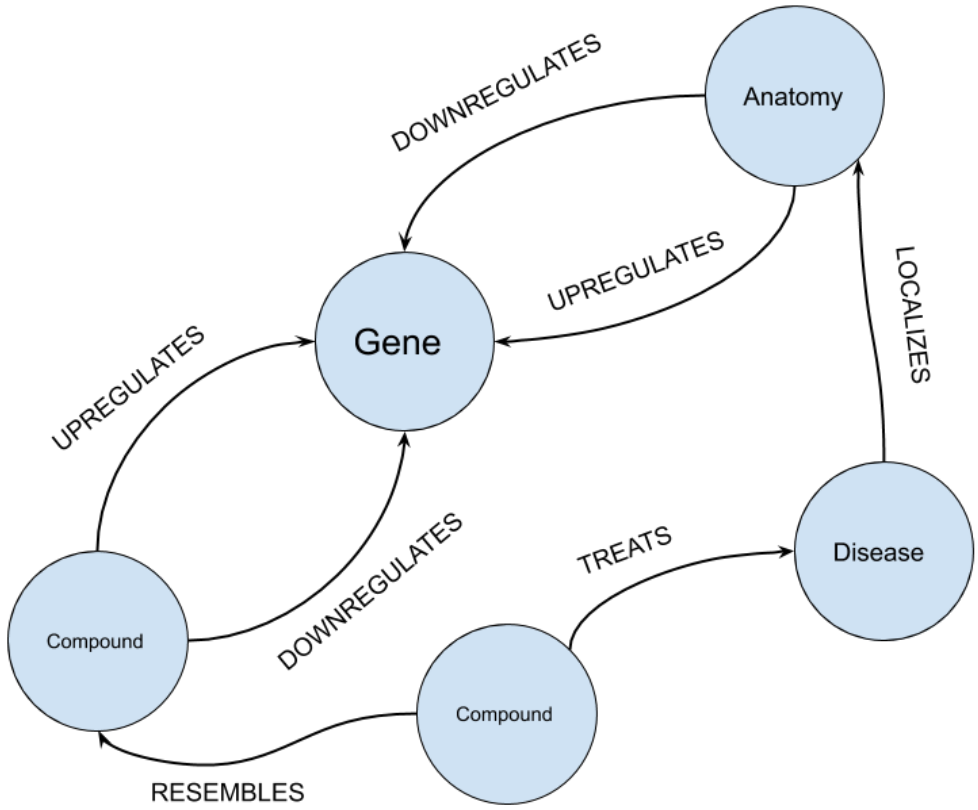
```
hetio_db["diseases"].bulk_write([
        UpdateOne({ "id" : $target }, {'$push': {'palliates_ids': $source }}),
        UpdateOne({ "id" : $target }, {'$push': {'treats_ids': $source }}),
        UpdateOne({ "id" : $source }, {'$push': {'localizes_ids': $target }}),
        ...
])
```

Potential Improvements:
There are many many lookups in the aggregate. It would be faster if all documents had an index on their `id` field.
The output does not look very good and could be made easier to read.

2. Supposed that a drug can treat a disease if the drug or its similar drugs (similarity>0.9) up-regulate/down-regulate a gene, but the location down-regulates/up-regulates the gene in an opposite direction where the disease occurs. Find all drugs that can treat a new disease (i.e. the missing edges between drug and disease excluding existing drugs). Obtain and output all drugs in a single query.

Database Used: Neo4j
- Neo4j is used because there are a complex amount of relationships that need to be mapped in order to answer the query. The lack of join statements in other NOSQL databases would make this very difficult.

Design Diagram:



Query:

```
MATCH (similar:Compound)-[:RESEMBLES]->(com:Compound)-[:TREATS]->(d:Disease{id: $disease_id})
WHERE NOT (similar)-[:TREATS]->(d)
AND (exists((similar)-[:UPREGULATES]->(:Gene)<-[:DOWNREGULATES]-(:Anatomy)<-[:LOCALIZES]-(d))
OR exists((similar)-[:DOWNREGULATES]->(:Gene)<-[:UPREGULATES]-(:Anatomy)<-[:LOCALIZES]-(d)))
RETURN (similar);
```

Import Nodes:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///nodes.tsv' AS row
FIELDTERMINATOR '\t'
FOREACH(ignoreMe IN CASE WHEN trim(row.kind) = "Disease" THEN [1] ELSE [] END | MERGE (p:Data:Disease{id:
row.id, name: row.name}))
FOREACH(ignoreMe IN CASE WHEN trim(row.kind) = "Gene" THEN [1] ELSE [] END | MERGE (p:Data:Gene{id: row.id,
name: row.name}))
FOREACH(ignoreMe IN CASE WHEN trim(row.kind) = "Anatomy" THEN [1] ELSE [] END | MERGE (p:Data:Anatomy{id:
row.id, name: row.name}))
FOREACH(ignoreMe IN CASE WHEN trim(row.kind) = "Compound" THEN [1] ELSE [] END | MERGE
(p:Data:Compound{id: row.id, name: row.name}));
```

Import Edges:

```
USING PERIODIC COMMIT 5000
LOAD CSV WITH HEADERS FROM 'file:///edges.tsv' AS row
FIELDTERMINATOR '\t'
WITH row WHERE row.metaedge IN ["CuG","AuG","CdG","AdG","CrC","CtD","DlA"]
MATCH (s:Data{id: row.source})
MATCH (t:Data{id: row.target})
```

```
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "CuG" THEN [1] ELSE [] END | MERGE
(s)-[:UPREGULATES]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "AuG" THEN [1] ELSE [] END | MERGE
(s)-[:UPREGULATES]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "CdG" THEN [1] ELSE [] END | MERGE
(s)-[:DOWNREGULATES]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "AdG" THEN [1] ELSE [] END | MERGE
(s)-[:DOWNREGULATES]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "CrC" THEN [1] ELSE [] END | MERGE
(s)<-[:RESEMBLES]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "CtD" THEN [1] ELSE [] END | MERGE
(s)-[:TREATS]->(t))
FOREACH(ignoreMe IN CASE WHEN row.metaedge = "DlA" THEN [1] ELSE [] END | MERGE
(s)-[:LOCALIZES]->(t));
```

Potential Improvements:
The import time is very slow (3 minutes). Data could be more efficiently preprocessed before import to speed up the process.

Instructions to run and use the program are in the README.md file.