

Assignment - 3

CSCI 5408 - Data Management, Warehousing, Analytics

Gitlab Repo: <https://git.cs.dal.ca/adimurthy/5408-assignment-3>

1.1 GCP Set-up and Apache Spark Installation

1.1.1 Pre-requisites: scala, java, and git

```
sudo apt install default-jdk
sudo apt install scala
sudo apt install git
```

1.1.2 Spark latest stable version:

```
wget
https://downloads.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
```

1.1.3 Extraction and move

```
tar xvf spark-3.2.0-bin-hadoop3.2.tgz
sudo mv spark-3.2.0-bin-hadoop3.2 /opt/spark
echo "export SPARK_HOME=/opt/spark" >> ~/.profile
echo "export SPARK_HOME=/opt/spark" >> ~/.profile
echo "export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin" >> ~/.profile
echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.profile
source ~/.profile
```

1.1.4 Update .profile

```
vi ~/.profile
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/usr/bin/python3
source ~/.profile
```

1.1.5 Starting the spark shell

```
spark-shell
```

1.1.6 Virtual machine created in GCP Compute Engine:

Google Cloud Platform

5408-assignment-2

Search products and resources

Compute Engine

spark-mapred...

EDIT

RESET

CREATE MACHINE IMAGE

OPERATIONS

HELP ASSISTANT

LEARN

Virtual machines

VM instances

Instance templates

Sole-tenant nodes

Machine images

TPUs

Committed use discounts

Migrate for Compute Engi...

Storage

Disks

Snapshots

Images

Instance groups

Instance groups

Marketplace

Release Notes

DETAILS

OBSERVABILITY

OS INFO

NEW

SCREENSHOT

SHOW MORE

Basic information

Name	spark-mapreduce
Instance Id	7111153002729206716
Description	None
Type	Instance
Status	Running
Creation time	Nov 11, 2021, 10:11:31 PM UTC-04:00
Zone	us-west4-b
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	None
Deletion protection	Disabled
Confidential VM service	Disabled
Preserved state size	0 GB

Machine configuration

Machine type	e2-medium
CPU platform	Intel Broadwell
Display device	Disabled
	Enable to use screen capturing and recording tools
GPUs	None

1.2 Starting the worker node and verifying the installation

1.2.2 Spark 3.2.0 version

```

root@ip-10-10-10-10:~# ssh -ss 0 0f156:03e:84:15:4a:C4:A8:17:D9:14:D6:f4
fc1:99:6b:17:29:47:ED:2E:63:22:AB:FE:A7:67:8B:1F:CF:71
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1021-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Nov 12 19:29:29 UTC 2021

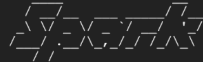
System load:  0.08             Processes:    107
Usage of /:   37.2% of 9.52GB  Users logged in: 0
Memory usage: 8%              IPV4 address for ens4: 10.182.0.6
Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

6 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

*** System restart required ***
Last login: Fri Nov 12 02:17:37 2021 from 35.235.242.210
root@ip-10-10-10-10:~# spark-shell
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.2.0.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/11/12 19:31:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://spark-mapreduce.us-west4-b.c.assignment-2-329321.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1636745479858).
Spark session available as 'spark'.
Welcome to

 version 3.2.0

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.11)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

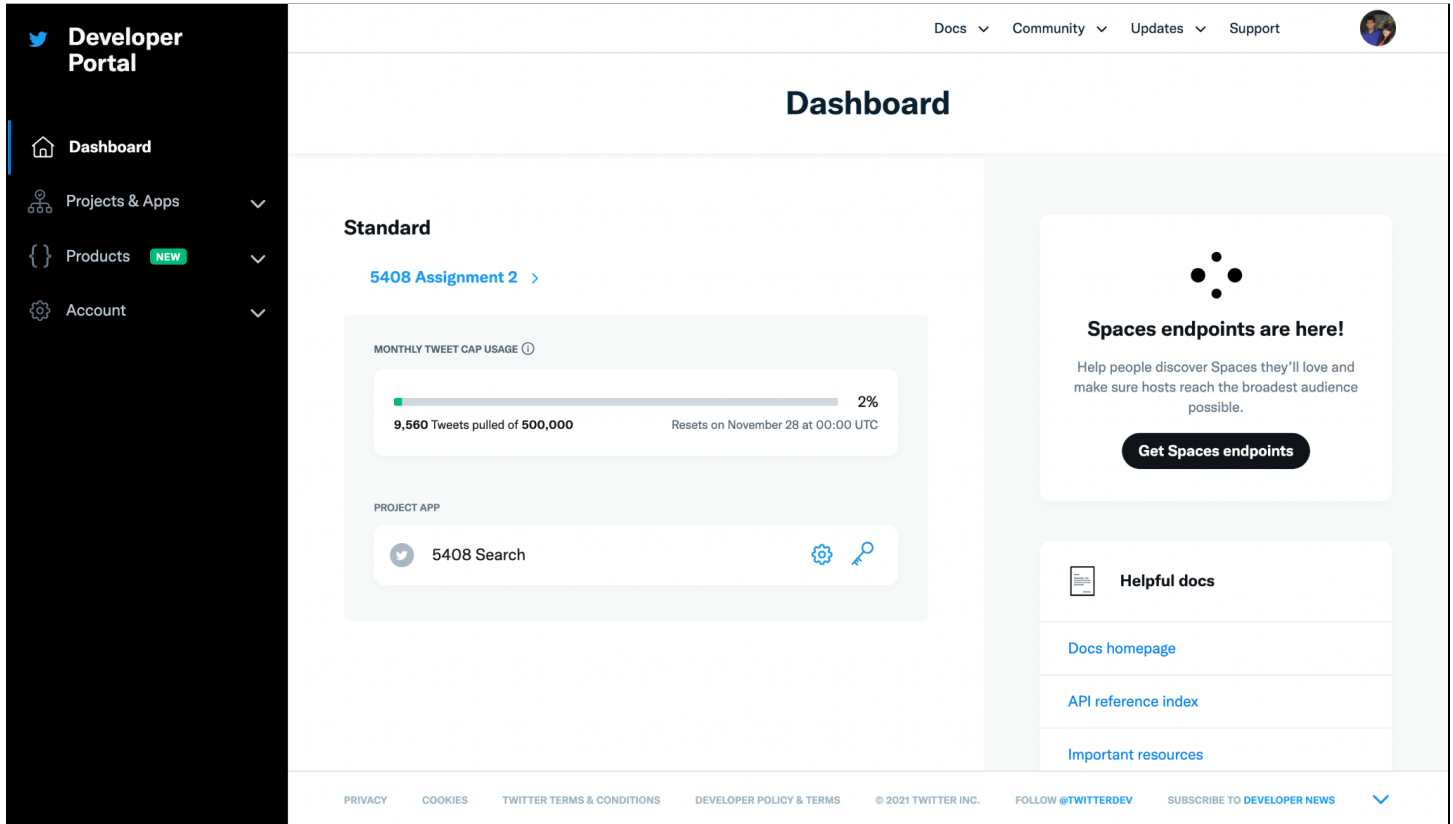
```

Furthermore, verified the working of simple operation in spark-shell, taking the example of word counter for a simpler use-case.

Note: I upgraded the instance from e2-micro to e2-medium, as e2-micro was very slow and CPU utilization peaked very often for simple operations.

2.1 Twitter Developer Account

The Twitter developer account activation took about a day. By the end of the assignment, below is the account dashboard of the Twitter developer account.



3.1 Twitter API and data format

I used the get-tweets-search-recent API, API reference:

<https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent>

3.2 Testing the API on postman before writing the java code

3.2.1 Curl Request (Example):

```
curl --location --request GET
'https://api.twitter.com/2/tweets/search/recent?query=canada&tweet.fields=author_id,entities,attachments,conversation_id,created_at,referenced_tweets&max_results=100' \
--header 'Authorization: Bearer <token>'
```

4.1 Java program to extract data from Twitter API

4.1.1 Gitlab:

<https://git.cs.dal.ca/adimurthy/5408-assignment-3/-/blob/main/src/main/java/com/example/Collector.java>

Used the HttpClient to make the API request to Twitter. Sensitive information such as the Bearer token is saved in the environmental variables in the GCP VM instance.

Entities to consider to parse the JSON to Java objects: [Tweet](#) and [Search Result](#).

Note: Twitter documentation mentions that the search keywords are not case-sensitive. Hence all the keywords are in lower case in the code-base.

4.1.2 Below is the main code section responsible for getting the tweets for the given keywords from the Twitter Recent tweets API ([Gitlab](#)):

```
for (String text: searchTexts) {
    for (int i = 0; i < 5; i++) {
        UriBuilder uriBuilder = new
UriBuilder("https://api.twitter.com/2/tweets/search/recent");
        ArrayList<NameValuePair> queryParameters;
        queryParameters = new ArrayList<>();
        queryParameters.add(new BasicNameValuePair("query", text));
        queryParameters.add(new BasicNameValuePair("tweet.fields",
"author_id,entities,created_at,possibly_sensitive,geo,lang,text,source"));
        queryParameters.add(new BasicNameValuePair("max_results", "100"));
        uriBuilder.addParameters(queryParameters);

        HttpGet httpGet = new HttpGet(uriBuilder.build());
        httpGet.setHeader("Authorization", String.format("Bearer %s",
bearerToken));
        httpGet.setHeader("Content-Type", "application/json");

        HttpResponse response = httpClient.execute(httpGet);
        HttpEntity entity = response.getEntity();

        tweetResponse = EntityUtils.toString(entity, "UTF-8");
        SearchResult result = mapper.readValue(tweetResponse,
SearchResult.class);

        MongoDB database = mongo.getDatabase("RawDb");

        for (Tweet tweet: result.getData()) {
            tweet.setKeyword(text);
            Document document = Document.parse(mapper.writeValueAsString(tweet));
```

```

        database.getCollection("tweets").insertOne(document);
    }

    if (Objects.nonNull(result.getMeta()) &&
Objects.nonNull(result.getMeta().getNextToken())) {
        queryParameters.add(new BasicNameValuePair("next_token",
result.getMeta().getNextToken()));
    }
}
}
}

```

5.1 Information Collected

While the API data is comprehensive, the information collection are:

1. ID (Unique Identifier) and Author ID
2. Text (tweet) and Referenced Tweets
3. Created At (Date format)
4. Mentions (account mentions in the tweets)
5. Geo and Organization.
6. Language
7. Source

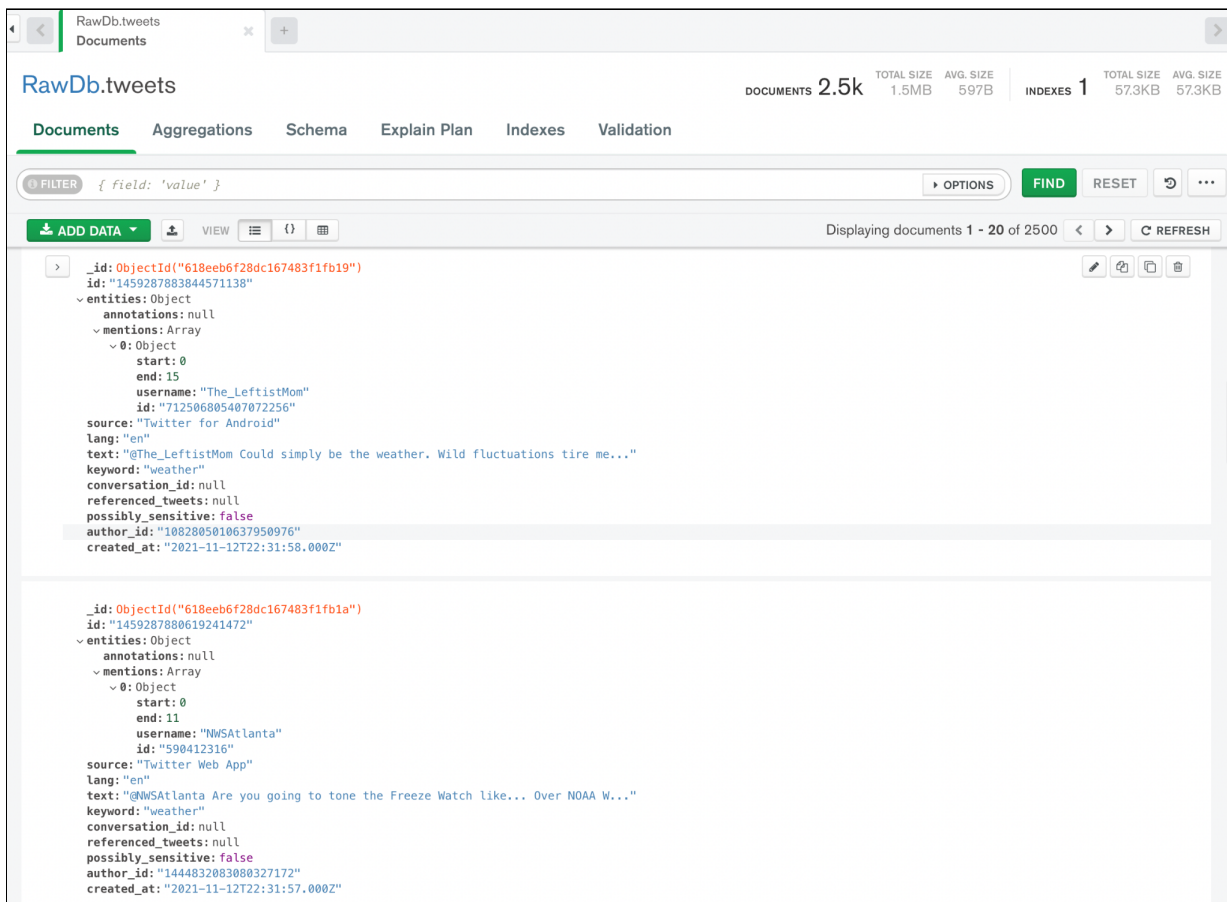
6.1 Raw Data storage

Since the firewall for the GCP instance was already configured to allow outbound traffic to any IP address, making a request to Twitter or connecting to MongoDB was hassle-free.

Instead of adding different collections within RawDb for different keywords, I have introduced a new field called "keyword," which represents the search word.

Note: Sensitive information such as the host and credentials are stored in environmental variables.

6.1.2 Illustration of data stored in MongoDB using Compass for the graphical user interface:



Note: The default value when the data for a field is not present in the API response is "null"

7.1 Process raw data:

7.1.1 Gitlab:

<https://git.cs.dal.ca/adimurthy/5408-assignment-3/-/blob/main/src/main/java/com/example/Processor.java>

7.1.2 Text Sanitization:

As mentioned earlier, the data in RawDb is not processed; the tweet text is sanitized to remove special characters, emoticons, extra spaces, and URLs using regex to check for the pattern and replace it with empty strings. Furthermore, the text processing is only for the text fields in the response, and no other fields are affected.

7.1.3 Processed data storage:

The sanitized data is stored in a new database called "ProcessedDb," under the collections "tweets," the schema of documents in the news is same as the tweets collection in RawDb. Below is a snapshot from MongoDB Compass illustrating the collections stored in ProcessedDb.

ProcessedDb.tweets

DOCUMENTS 2.5k TOTAL SIZE 1.4MB AVG. SIZE 573B INDEXES 1 TOTAL SIZE 4.1KB AVG. SIZE 4.1KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' }

ADD DATA VIEW

Displaying documents 1 - 20 of 2500 REFRESH

```

{
  "_id": ObjectId("618eee34d6732a6db86784f3"),
  "id": "1459287897597784075",
  "entities": {
    "annotations": null,
    "mentions": [
      {
        "start": 0,
        "end": 14,
        "username": "MagisterLatro",
        "id": "1336796471379677185"
      }
    ]
  },
  "source": "Twitter for Android",
  "lang": "en",
  "text": "MagisterLatro We get snowstorms sometimes and the weather can get cold...",
  "keyword": "weather",
  "conversation_id": null,
  "referenced_tweets": null,
  "possibly_sensitive": false,
  "author_id": "1298655648356470785",
  "created_at": "2021-11-12T22:32:01.000Z"
}

{
  "_id": ObjectId("618eee34d6732a6db86784f4"),
  "id": "1459287891612344320",
  "entities": {
    "annotations": [
      {
        "start": 78,
        "end": 86,
        "probability": 0.9029,
        "type": "Place",
        "normalized_text": "Sao Paulo"
      }
    ]
  },
  "mentions": null,
  "source": "Twitter Web App",
  "lang": "en",
  "text": "F1 BrazilGP WEATHER UPDATE INTERLAGOS Moisture flow starts to decrease...",
  "keyword": "weather",
  "conversation_id": null,
  "referenced_tweets": null,
  "possibly_sensitive": false,
  "author_id": "608601783"
}

```

9.1 Algorithms for data sanitization and transformation:

9.1.1 Gitlab:

<https://git.cs.dal.ca/adimurthy/5408-assignment-3/-/blob/main/src/main/java/com/example/Processor.java>

9.1.2 Screenshot of the above code

The screenshot only has the process method responsible for fetching data from RawDb, processing tweets and store in ProcessedDb.

```

public static void process() throws IOException {
    MongoClient mongo = new MongoClient(System.getenv("MONGO_HOST"), 27017);
    List<Tweet> tweets = new ArrayList<>();
    ObjectMapper mapper = new ObjectMapper();

    MongoDBDatabase rawDatabase = mongo.getDatabase("RawDb");
    MongoDBDatabase processedDatabase = mongo.getDatabase("ProcessedDb");
}

```

```

MongoCollection<Document> collection = rawDatabase.getCollection("tweets");

MongoCursor<Document> cursor = collection.find().iterator();
while (cursor.hasNext()) {
    tweets.add(mapper.readValue(cursor.next().toJson(), Tweet.class));
}

for (Tweet tweet: tweets) {
    String text = sanitize(tweet.getText());
    tweet.setText(text);

    Document document = Document.parse(mapper.writeValueAsString(tweet));
    processedDatabase.getCollection("tweets").insertOne(document);
}
}

private static String sanitize(String text) {
    text = removeUrl(text);
    text = removeSpecialCharacters(text);
    text = removeSpaces(text);
    return text;
}

```

10.1 Text extraction between tags, processing, and storage:

10.1.1 Gitlab:

<https://git.cs.dal.ca/adimurthy/5408-assignment-3/-/blob/main/src/main/java/com/example/Reuter.java>

10.1.2 Text extraction between tags:

The method "getTags" takes the string text and the tag. Returns the list of strings between the tags, and it's a list if there are multiple tags in the input string.

```

private static List<String> getTags(final String text, final String tag) {
    Pattern regex = Pattern.compile(String.format("<%s.*?>(.*?)</%s>", tag, tag),
    Pattern.DOTALL);
    final List<String> matches = new ArrayList<>();
    final Matcher matcher = regex.matcher(text);
    while (matcher.find()) {
        matches.add(matcher.group(1));
    }
    return matches;
}

```


10.1.3 Code for text extraction and storage:

```
private static void store(String filename) throws IOException {
    List<String> reutersMatches;
    InputStream inputStream =
    Reuter.class.getClassLoader().getResourceAsStream(filename);
    ObjectMapper mapper = new ObjectMapper();

    MongoClient mongo = new MongoClient(System.getenv("MONGO_HOST") , 27017);
    MongoDB database = mongo.getDatabase("ReutersDb");

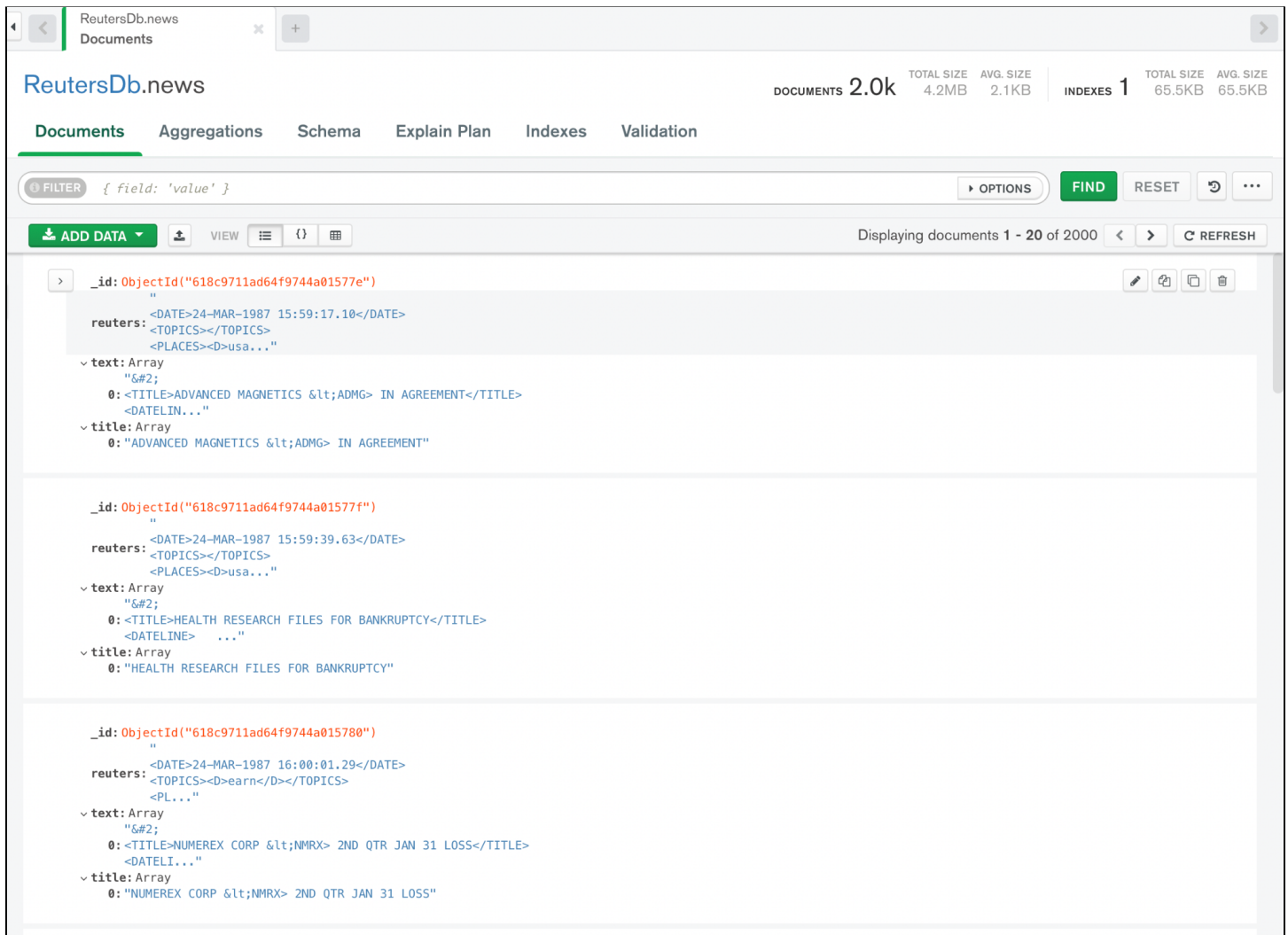
    String text = new Scanner(inputStream).useDelimiter("\\A").next();

    reutersMatches = getTags(text, "REUTERS");

    for (String reutersMatch: reutersMatches) {
        News news = News.builder()
            .reuters(reutersMatch)
            .build();
        List<String> textMatches = getTags(reutersMatch, "TEXT");
        news.setText(textMatches);
        for (String textMatch: textMatches) {
            List<String> titleMatches = getTags(textMatch, "TITLE");
            news.setTitle(titleMatches);
        }
        Document document = Document.parse(mapper.writeValueAsString(news));
        database.getCollection("news").insertOne(document);
    }
}
```

10.1 Reuters collection in Mongo DB

Text and title are lists for extensibility, if there are multiple tags within <Reuters></Reuters>



The screenshot shows the MongoDB Compass interface for the 'ReutersDb.news' collection. The top bar indicates 2.0k documents, 4.2MB total size, and 2.1KB average size. The 'Documents' tab is active, showing a list of documents. The first document is expanded, showing its structure:

```
{
  "_id": ObjectId("618c9711ad64f9744a01577e"),
  "reuters": {
    "<DATE>24-MAR-1987 15:59:17.10</DATE>": "<TOPICS></TOPICS>",
    "<PLACES><D>usa...": ""
  },
  "text": Array [
    "&#2;"
  ],
  "title": Array [
    "<TITLE>ADVANCED MAGNETICS &lt;ADMG> IN AGREEMENT</TITLE>"
  ],
  "dateLine": ""
}
```

11.1 MapReduce Frequency Count of keywords:

11.1.1 Gitlab:

<https://git.cs.dal.ca/adimurthy/5408-assignment-3/-/blob/main/src/main/java/com/example/Generator.java>

11.2 File Writer

The data stored in ReutersDb and ProcessedDb are sanitized and stored in files to make it easier for the MapReduce program to access the data to find the frequency of the keywords.

```
MongoClient mongo = new MongoClient(System.getenv(System.getenv("MONGO_HOST")),
27017);
ObjectMapper mapper = new ObjectMapper();

List<Tweet> tweets = new ArrayList<>();
List<News> news = new ArrayList<>();
```

```

BufferedWriter tweetsFile = new BufferedWriter(new FileWriter("tweets.json"));
BufferedWriter newsFile = new BufferedWriter(new FileWriter("news.json"));

MongoDatabase ReutersDatabase = mongo.getDatabase("ReutersDb");
MongoDatabase processedDatabase = mongo.getDatabase("ProcessedDb");

MongoCollection<Document> newsCollection =
ReutersDatabase.getCollection("news");
MongoCollection<Document> tweetCollection =
processedDatabase.getCollection("tweets");

MongoCursor<Document> newsCursor = newsCollection.find().iterator();
while (newsCursor.hasNext()) {
    News mapperNews = mapper.readValue(newsCursor.next().toJson(), News.class);
    newsFile.write(mapperNews.getReuters().toLowerCase().concat(" "));
}

MongoCursor<Document> tweetCursor = tweetCollection.find().iterator();
while (tweetCursor.hasNext()) {
    Tweet mapperTweet = mapper.readValue(tweetCursor.next().toJson(),
Tweet.class);
    tweetsFile.write(mapperTweet.getText().concat(" "));
}

```

11.3 MadReduce Program

11.3.1 Dependency:

```

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.13</artifactId>
  <version>3.2.0</version>
</dependency>

```

11.3.2 Program to get the count of words (Frequency):

The function takes the filename and the list of keywords and returns a list of tuples, where each tuple represents a string having the full keyword or a sub-string along with its frequency count.

```

private static List<Tuple2<String, Integer>> getCount(String filename,
List<String> keywords) {
    SparkConf conf = new SparkConf().setMaster("local").setAppName("wordCount");
    JavaSparkContext sc = new JavaSparkContext(conf);

```

```

JavaRDD<String> file = sc.textFile(filename);

JavaRDD<String> words = file
    .flatMap(word -> Arrays.asList(word.split(" ")).iterator())
    .filter(word -> keywords.stream().anyMatch(keyword ->
word.toLowerCase().contains(keyword)))
    .filter(word -> !word.isEmpty());

JavaPairRDD<String, Integer> wordMap_to_pair = words
    .mapToPair(f -> new Tuple2<String, Integer>(f, 1));

JavaPairRDD<String, Integer> frequencyCounters =
wordMap_to_pair.reduceByKey((a,b) -> a + b);
System.out.println(frequencyCounters.collect());

return frequencyCounters.collect();
}

```

11.4 Frequency Count:

The frequency count mentioned below is an aggregation of the count of both the collections in ProcessedDb and ReutersDb.

The count of each keyword:

1. rain: 518
2. education: 315
3. canada: 630
4. snow: 33
5. indoor: 5
6. ice: 1350
7. cold: 70
8. hot: 84
9. flu: 64

12.1 Explore and graph creation

12.1.1 Reference for hierarchy:

<https://www.worldatlas.com/geography/capital-cities-of-canada-s-provinces-territories.html>

12.2 Entities:

1.2.1 Provinces:

```
CREATE (n:Province {name: 'Alberta', abbreviation: 'AB'});
CREATE (n:Province {name: 'British Columbia', abbreviation: 'BC'});
CREATE (n:Province {name: 'Manitoba', abbreviation: 'MB'});
CREATE (n:Province {name: 'New Brunswick', abbreviation: 'NB'});
CREATE (n:Province {name: 'Newfoundland and Labrador', abbreviation: 'NL'});
CREATE (n:Province {name: 'Ontario', abbreviation: 'ON'});
CREATE (:Province {name: 'Nova Scotia', abbreviation: 'NS'})
CREATE (n:Province {name: 'Prince Edward Island', abbreviation: 'PE'});
CREATE (n:Province {name: 'Quebec', abbreviation: 'QC'});
CREATE (n:Province {name: 'Saskatchewan', abbreviation: 'SK'});
```

1.2.2 Territories

```
CREATE (n: Territory {name: 'Yukon', abbreviation: 'YT'});
CREATE (n: Territory {name: 'Nunavut', abbreviation: 'NU'});
CREATE (n: Territory {name: 'Northwest Territories', abbreviation: 'NT'});
```

1.2.3 Cities

```
CREATE (n:City {name: 'Edmonton'});
CREATE (n:City {name: 'Victoria'});
CREATE (n:City {name: 'Winnipeg'});
CREATE (n:City {name: 'Fredericton'});
CREATE (n:City {name: " St. John's"});
CREATE (n:City {name: " Halifax"});
CREATE (n:City {name: " Toronto"});
CREATE (n:City {name: " Charlottetown"});
CREATE (n:City {name: " Quebec City"});
CREATE (n:City {name: " Regina"});
CREATE (n:City {name: " Whitehorse"});
CREATE (n:City {name: " Iqaluit"});
CREATE (n:City {name: " Yellowknife"});
CREATE (n:City {name: "Calgary"});
CREATE (n:City {name: " Vancouver"});
CREATE (n:City {name: " Saint John"});
CREATE (n:City {name: " Montreal"});
CREATE (n:City {name: " Saskatoon"});
```

12.3 Relationships:

12.3.1 Neighbors:

Alberta

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Alberta' AND b.name = 'Saskatchewan' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Alberta' AND b.name = 'British Columbia' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Alberta' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Alberta' AND b.name = 'Yukon' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

British Columbia

```
MATCH
  (a:Province),
  (b:Province)
WHERE a.name = 'British Columbia' AND b.name = 'Alberta' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b:Territory)
WHERE a.name = 'British Columbia' AND b.name = 'Yukon' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```



```

MATCH
  (a:Province),
  (b:Territory)
WHERE a.name = 'British Columbia' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Quebec:

```

MATCH
  (a:Province),
  (b:Province)
WHERE a.name = 'Quebec' AND b.name = 'New Brunswick' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a:Province),
  (b:Province)
WHERE a.name = 'Quebec' AND b.name = 'Newfoundland and Labrador' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a:Province),
  (b:Province)
WHERE a.name = 'Quebec' AND b.name = 'Ontario' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Nova Scotia

```

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Nova Scotia' AND b.name = 'Newfoundland and Labrador' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Nova Scotia' AND b.name = 'Prince Edward Island' CREATE

```

```
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

New Brunswick

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'New Brunswick' AND b.name = 'Prince Edward Island' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'New Brunswick' AND b.name = 'Nova Scotia' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'New Brunswick' AND b.name = 'Quebec' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

Newfoundland and Labrador

```
MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Newfoundland and Labrador' AND b.name = 'Quebec' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

Prince Edward Island

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Prince Edward Island' AND b.name = 'Nova Scotia' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Prince Edward Island' AND b.name = 'New Brunswick' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Northwest Territories

```

MATCH
  (a: Territory),
  (b: Territory)
WHERE a.name = 'Northwest Territories' AND b.name = 'Yukon' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a: Territory),
  (b: Territory)
WHERE a.name = 'Northwest Territories' AND b.name = 'Nunavut' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Northwest Territories' AND b.name = 'British Columbia' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Northwest Territories' AND b.name = 'Alberta' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Northwest Territories' AND b.name = 'Saskatchewan' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

```
MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Northwest Territories' AND b.name = 'Manitoba' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

Manitoba

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Manitoba' AND b.name = 'Ontario' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```
MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Manitoba' AND b.name = 'Saskatchewan' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```
MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Manitoba' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

```
MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Manitoba' AND b.name = 'Nunavut' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);
```

Nunavut

```
MATCH
  (a: Territory),
  (b: Territory)
WHERE a.name = 'Nunavut' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
```

```

RETURN type(r);

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Nunavut' AND b.name = 'Manitoba' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Nunavut' AND b.name = 'Saskatchewan' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Ontario

```

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Ontario' AND b.name = 'Quebec' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Ontario' AND b.name = 'Manitoba' CREATE (a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Saskatchewan

```

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Saskatchewan' AND b.name = 'Manitoba' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Province)
WHERE a.name = 'Saskatchewan' AND b.name = 'Alberta' CREATE
(a)-[r:NEIGHBOUR]->(b)

```

```

RETURN type(r);

MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Saskatchewan' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a:Province),
  (b: Territory)
WHERE a.name = 'Saskatchewan' AND b.name = 'Nunavut' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

Yukon

```

MATCH
  (a: Territory),
  (b: Province)
WHERE a.name = 'Yukon' AND b.name = 'British Columbia' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

MATCH
  (a: Territory),
  (b: Territory)
WHERE a.name = 'Yukon' AND b.name = 'Northwest Territories' CREATE
(a)-[r:NEIGHBOUR]->(b)
RETURN type(r);

```

12.3.2 Largest:

```

MATCH
  (a:City),
  (b: Province)
WHERE a.name = 'Saskatoon' AND b.name = 'Saskatchewan' CREATE
(a)-[r:LARGEST_IN]->(b)
RETURN type(r);

MATCH
  (a:City),

```



```

    (b: Province)
WHERE a.name = ' Montreal' AND b.name = 'Quebec' CREATE (a)-[r:LARGEST_IN]->(b)
RETURN type(r);

MATCH
    (a:City),
    (b: Province)
WHERE a.name = ' Vancouver' AND b.name = 'British Columbia' CREATE
(a)-[r:LARGEST_IN]->(b)
RETURN type(r);

MATCH
    (a:City),
    (b: Province)
WHERE a.name = 'Calgary' AND b.name = 'Alberta' CREATE (a)-[r:LARGEST_IN]->(b)
RETURN type(r);

MATCH
    (a:City),
    (b: Province)
WHERE a.name = ' Saint John' AND b.name = 'New Brunswick' CREATE
(a)-[r:LARGEST_IN]->(b)
RETURN type(r);

```

12.3.3 Capital:

```

MATCH
    (a:City),
    (b: Province)
WHERE a.name = 'Winnipeg' AND b.name = 'Manitoba'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);

MATCH
    (a:City),
    (b: Province)
WHERE a.name = 'Edmonton' AND b.name = 'Alberta'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);

MATCH
    (a:City),
    (b: Territory)

```

```
WHERE a.name = ' Whitehorse' AND b.name = 'Yukon'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = 'Fredericton' AND b.name = 'New Brunswick'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = " St. John's" AND b.name = 'Newfoundland and Labrador'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = 'Victoria'AND b.name = 'British Columbia'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = ' Halifax' AND b.name = 'Nova Scotia'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = ' Quebec City' AND b.name = 'Quebec'  
CREATE (a)-[r:CAPITAL]->(b)  
RETURN type(r);
```

```
MATCH  
  (a:City),  
  (b: Province)  
WHERE a.name = ' Regina' AND b.name = 'Saskatchewan'
```

```
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);
```

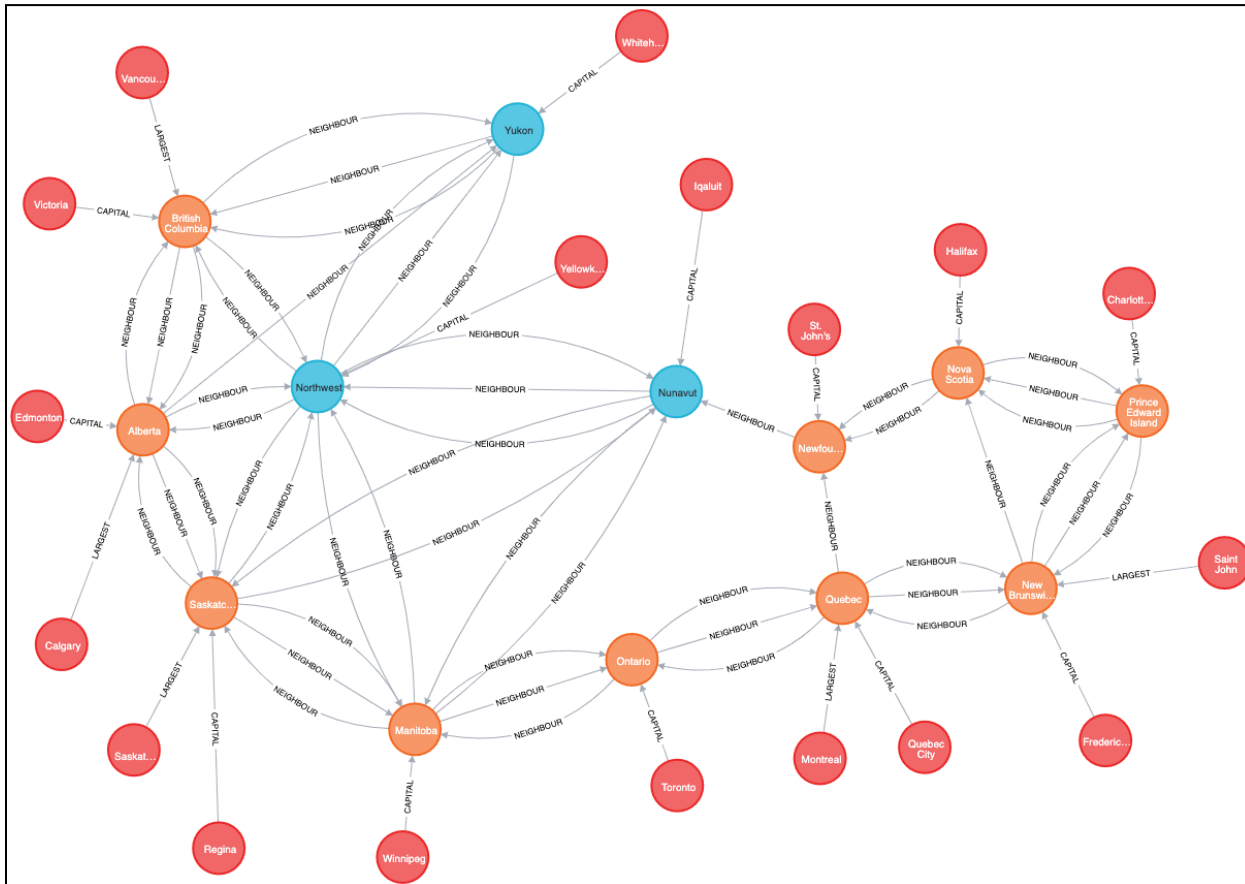
```
MATCH
  (a:City),
  (b: Province)
WHERE a.name = ' Toronto' AND b.name = 'Ontario'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);
```

```
MATCH
  (a:City),
  (b: Province)
WHERE a.name = ' Charlottetown' AND b.name = 'Prince Edward Island'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);
```

```
MATCH
  (a:City),
  (b: Territory)
WHERE a.name = ' Iqaluit' AND b.name = 'Nunavut'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);
```

```
MATCH
  (a:City),
  (b: Territory)
WHERE a.name = ' Yellowknife' AND b.name = 'Northwest Territories'
CREATE (a)-[r:CAPITAL]->(b)
RETURN type(r);
```

12.4 Graph:



13.1 General References:

Google Cloud Platform. Accessed on: Nov 3, 2021. [Online]. Available: <https://cloud.google.com/gcp>

Virtual machine instances. Accessed on: Nov 3, 2021. [Online]. Available: <https://cloud.google.com/compute/docs/instances>

Java Pattern and Regex. Accessed on: Nov 5, 2021. [Online]. Available: <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>

Maven Apache Spark. Accessed on: Nov 6, 2021. [Online]. Available: <https://mvnrepository.com/artifact/org.apache.spark>

MongoDB java client. Accessed on: Nov 6, 2021. [Online]. Available: <https://docs.mongodb.com/drivers/java-drivers/>

Maven MongoDB java client. Accessed on: Nov 6, 2021. [Online]. Available: <https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver>

MongoDB Compass. Accessed on: Nov 6, 2021. [Online]. Available: <https://www.mongodb.com/products/compass>

Twitter Recent Tweets API. Accessed on: Nov 7, 2021. [Online]. Available: <https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent>

Graph database neo4j. Accessed on: Nov 8, 2021. [Online]. Available: <https://neo4j.com/>

Capital Cities Of Canada's Provinces/Territories. Accessed on: Nov 8, 2021. [Online]. Available: <https://www.worldatlas.com/geography/capital-cities-of-canada-s-provinces-territories.html>