CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

Assignment - 2

Gitlab Link: https://git.cs.dal.ca/jspatel/csci5408_s24_b00982253_jay_patel

References

Problem 1A

Algorithm

Initialize MongoDB Connection:

- Create a MongoClient instance using a connection URI.
- Connect to the ReuterDb database and the News collection.

Read and Parse the News File:

- Create an instance of FileOperations using a singleton pattern.
- Read the raw data from the specified file (reut2-014.sgm) into a StringBuilder.

Extract and Clean Data:

- Use regular expressions to extract individual news articles (<REUTERS> tags) from the raw data.
- For each extracted article, extract the TITLE and BODY tags.
- Clean the extracted TITLE and BODY content by removing special characters and trimming whitespace.

Convert Data to MongoDB Documents:

- Convert the cleaned TITLE and BODY content into News objects.
- Create a list of Document objects from the News objects.

Insert Data into MongoDB:

- Insert the list of Document objects into the News collection in MongoDB.
- Print a success message upon successful insertion.

The main method reads news articles from the specified file (reut2-014.sgm) using the FileOperations class, which implements a singleton pattern to ensure only one instance is created.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

6 usages
public class FileOperations {

    4 usages
    private static FileOperations fileOperation;

    1 usage
    public static FileOperations getInstance() {
        if (fileOperation == null) {
            //Use Synchronized, so that in case of multithreading or multiple user create object at the same time
            // only one thread goes inside this block.
            synchronized (FileOperations.class) {
                if (fileOperation == null){
                    //Create one time object.
                    fileOperation = new FileOperations();
                }
            }
        }
        return fileOperation;
    }
```

Figure 1: Singleton FileOperations class.

The getNews method in FileOperations reads the file content into a StringBuilder and then uses regular expressions to extract the content within <REUTERS> tags. Each extracted article's TITLE and BODY tags are further parsed, cleaned, and stored in News objects. Once the articles are parsed and cleaned, the insertDataToDB method converts these News objects into MongoDB Document objects and inserts them into the News collection in the ReuterDb database. The design encapsulates file reading, data extraction, cleaning, and database insertion in a structured and efficient manner.

FlowChart

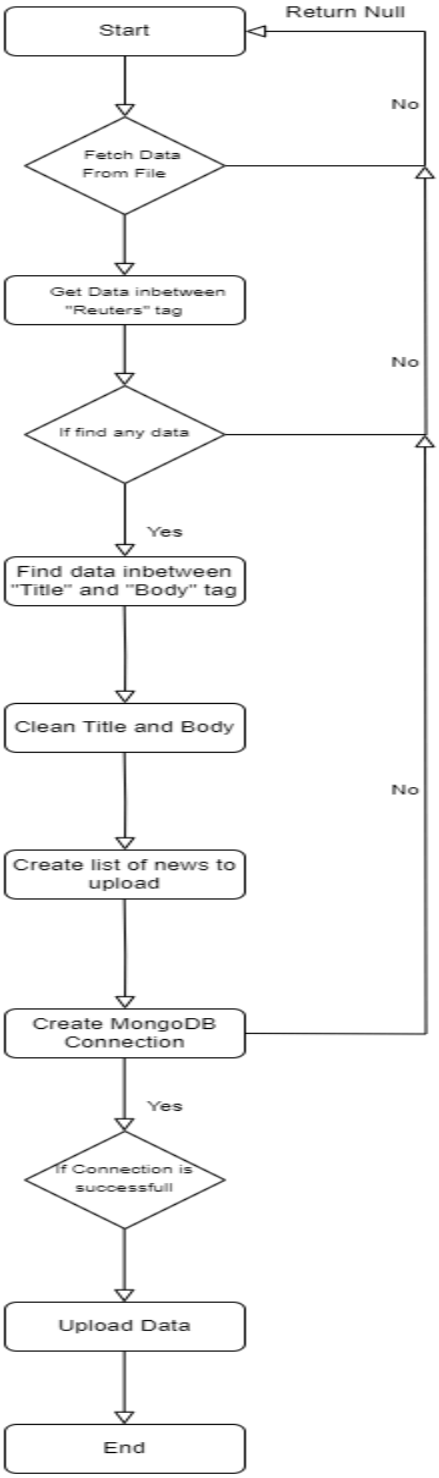

Figure 2: FlowChart of Problem 1A.

Data Cleaning/Transformation

1. Extraction of Relevant Data
    - I used <REUTERS.*?>(.*?)</REUTERS> regex pattern matches the entire content enclosed within <REUTERS> and </REUTERS> tags. The .*? makes the match non-greedy, ensuring that it captures the shortest possible string that fits the pattern.

```
private List<String> getInfoFromRawData(StringBuilder content) {
    List<String> results = new ArrayList<>();
    String regex = "<REUTERS.*?>(.*?)</REUTERS>";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(content);
    while (matcher.find()) {
        results.add(matcher.group(1));
    }
    return results;
}
```

Figure 3: Extract data between "REUTERS" tag.

2. Parsing Title and Body
    - I used <TITLE.*?>(.*?)</TITLE> this pattern extracts content between <TITLE> and </TITLE> tags, similarly using .*? for a non-greedy match.
    - Moreover, I used the pattern <BODY.*?>(.*?)</BODY to extract content between <BODY> and </BODY> tags, similarly using .*? for a non-greedy match.

```
private String getTitleAndBody(String content, String tagName) {
    String regex = "<" + tagName + ".*?>(.*?)</" + tagName + ">";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(content);
    if (matcher.find()) {
        return matcher.group(1);
    }
    return null;
}
```

Figure 4: Extract title and body from each news.

3. Cleaning Data
    - Removing Special Characters:
        - ➔ Used this pattern [^a-zA-Z0-9\\s] for matching any character that is not a letter, digit, or whitespace. It removes all such characters to ensure that only alphanumeric characters and spaces are retained.

- Replacing HTML Entities:
  - ➔ < and > is replaced with an empty string.
  - ➔ This step removes specific HTML entities that could appear in the text.
- Trimming Whitespace:
  - ➔ I used this pattern \\s+ which matches one or more whitespace characters (spaces, tabs, newlines) and is replaced with a single space to consolidate multiple spaces into one.
  - ➔ I also used trim() to remove white spaces at from both ends.

```
private String cleanTitleAndBody(String content) {
    if(content == null)
        return content;
    return content.replaceAll( regex: "[^a-zA-Z0-9\\s]", replacement: "")
            .replace( target: "&lt;", replacement: "").replace( target: ">", replacement: "")
            .replaceAll( regex: "\\s+", replacement: " ").trim();
}
```

Figure 5:Remove special characters and unnecessary spaces.

4. Transforming Data into Objects
   - Conversion to News Objects:
     - ➔ The cleaned TITLE and BODY strings are used to create News objects. Each News object holds the title and body of the news article in a structured format.

```
private List<News> filterData(List<String> info){

    List<News> response = new ArrayList<>();

    for (String currInfo : info) {
        News currNew = new News();
        String title = getTitleAndBody(currInfo, tagName: "TITLE");
        String body = getTitleAndBody(currInfo, tagName: "BODY");
        if (title != null || body != null) {
            currNew.setTitle(cleanTitleAndBody(title));
            currNew.setBody(cleanTitleAndBody(body));
            response.add(currNew);
        }
    }

    return response;
}
```

Figure 6: Create News from filtered data.

5. Preparation for MongoDB Insertion
    - Conversion to MongoDB Document Objects:
    - Document Creation:
        - ➔ Each news object is converted into a document with fields, such as title and body. This transformation prepares the data for insertion into the MongoDB collection.

```
    private static void insertDataToDB(List<News> response){

        List<Document> data = new ArrayList<>();

        for (News currNew : response){
            Document doc = new Document("title", currNew.getTitle())
                    .append("body", currNew.getBody());
            data.add(doc);
        }

        collection.insertMany(data);
        System.out.println("Data Inserted Successfully To MongoDB Database!!");
    }

}
```

Figure 7: Insert data into MongoDB database.

MongoDB Connection

```
public class ReutReader {

    2 usages
    private static MongoClient mongoClient;
    2 usages
    private static MongoCollection<Document> collection;
    2 usages
    private static FileOperations fileOperation;


    public static void main(String[] args) {
        fileOperation = FileOperations.getInstance();
        List<News> response = fileOperation.getNews( filepath: "reut2-014.sgm");

        ConnectMongoDB();
        MongoDatabase database = mongoClient.getDatabase( databaseName: "ReuterDb");
        collection = database.getCollection( s: "News");
        insertDataToDB(response);
    }

    1 usage
    private static void ConnectMongoDB(){
        MongoClientURI uri = new MongoClientURI("mongodb+srv://root:root@default-cluster.xn53b0c.mongodb.net/?retryWrites=true&w=majority&appName=default-cluster");
        mongoClient = new MongoClient(uri);
    }
```

Figure 8: MongoDB database connection string and collection name.

- The `ConnectMongoDB` method establishes a connection to a MongoDB database using the `MongoClient` class with a connection URI. This URI specifies the MongoDB cluster's address, authentication credentials, and connection options.

Output

- Here I create database with name “ReuterDb” and collection name is “News”.

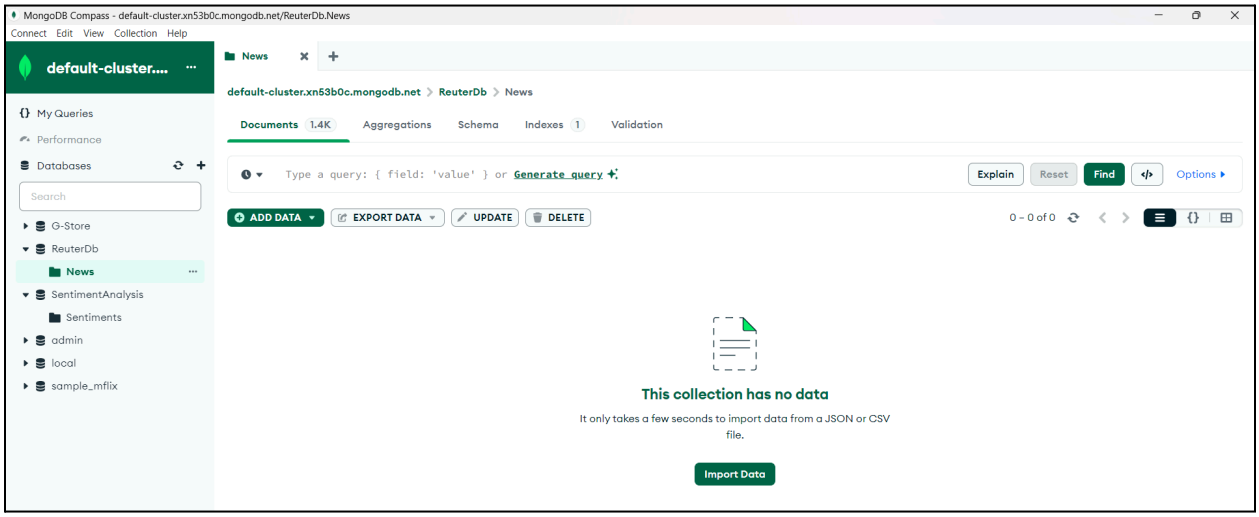

Figure 9: Before we run Java program.

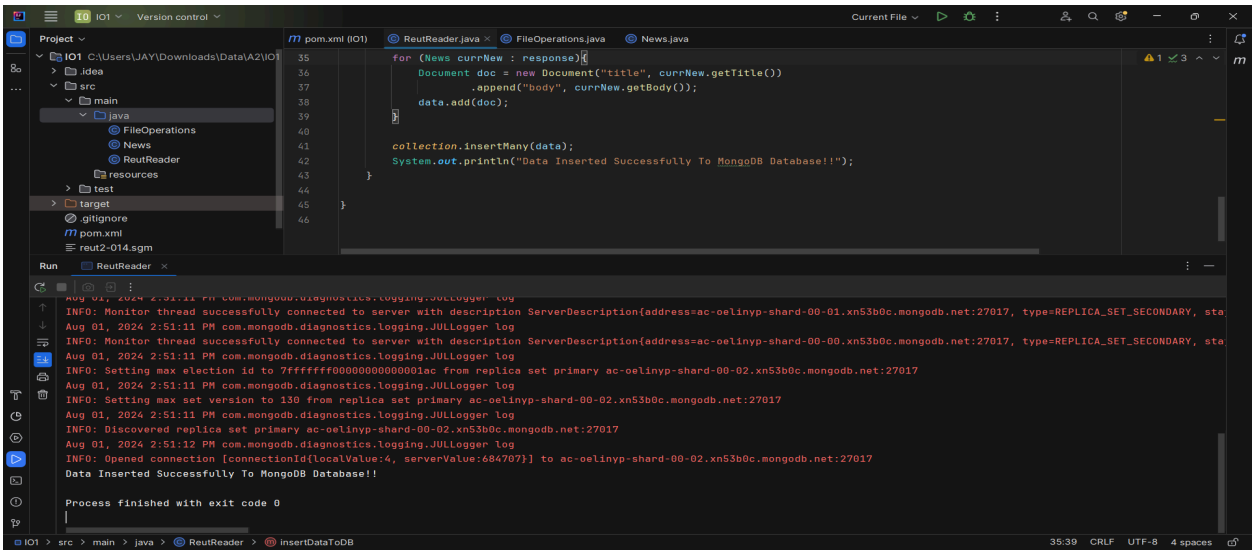

Figure 10: Run Java Program.

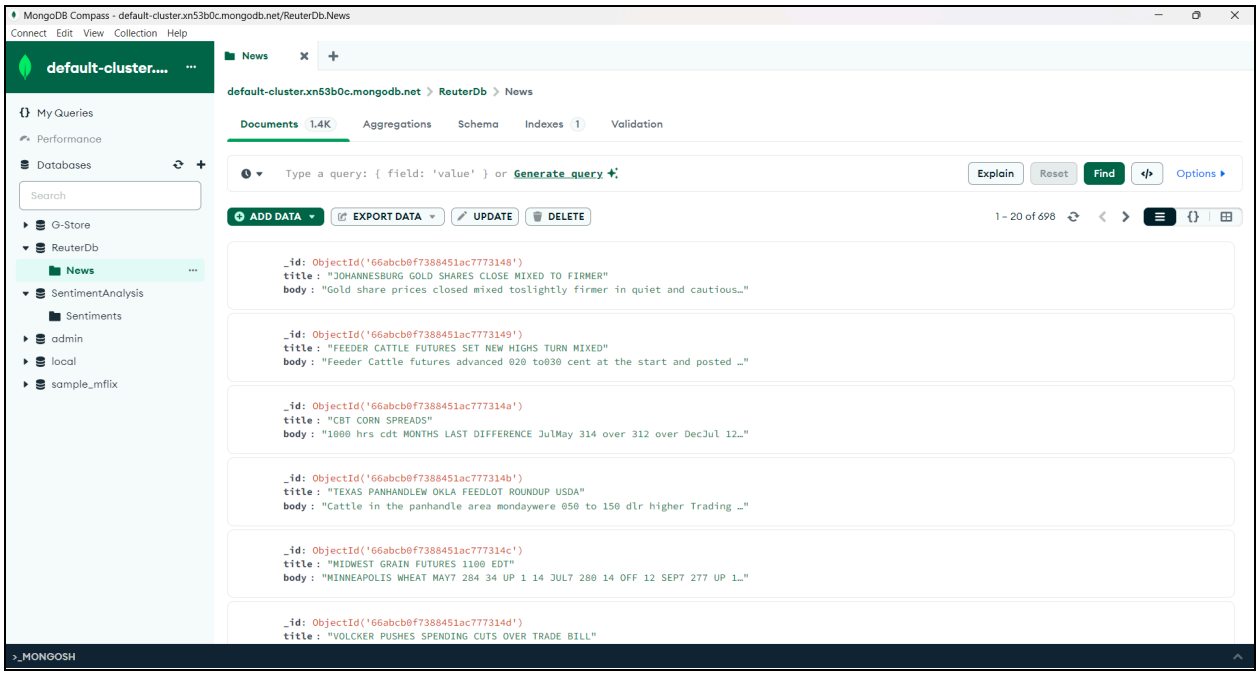

Figure 11: After successfully run the java program, data on MongoDB

Problem 1B

Algorithm

Initialize Spark Session

- Create a SparkSession instance with the application name "A2".

```
String appName = "A2";
SparkSession sparkSession = SparkSession.builder().appName(appName).getOrCreate();
```

Figure 12: Start spark session with "A2" app name.

Read Input Data

- Use the FileOperations class to read the main data file ("reut2-014.sgm") and the stop words file ("StopWords.txt") using the getData and getStopWords methods, respectively.
- Collect the stop words into a List<String>.

```
1 usage
public JavaRDD<String> getData(SparkSession sparkSession, String filePath){
    Dataset<String> fileData = sparkSession.read().option("multiline", false).textFile(filePath);
    //Print File to Console
    System.out.println("Content of File: ");
    fileData.show();
    return fileData.javaRDD();
}

1 usage
public JavaRDD<String> getStopWords(SparkSession sparkSession, String filePath){
    Dataset<String> fileData = sparkSession.read().option("multiline", false).textFile(filePath);
    //Print Stop Words to Console
    System.out.println("Stop Words: ");
    fileData.show();
    return fileData.javaRDD();
}
```

Figure 13: Fetch data from "reut2-014.sgm", and "StopWords.txt".

Clean and Tokenize Text

- Clean Text:
  - ➔ For each line in the main data file, replace all non-alphabetic characters (except spaces) with a space.
  - ➔ Convert the text to lowercase.
  - ➔ Trim leading and trailing whitespace.

➔ Split the cleaned text into individual words by whitespace.

```
JavaRDD<String> cleanedText = news.flatMap(line -> Arrays.asList(
        line.replaceAll( regex: "[^a-zA-Z\\s]", replacement: " ")
                .toLowerCase(Locale.ROOT)
                .trim()
                .split( regex: "\\s+")
).iterator());
```

Figure 14: Remove unnecessary spaces and convert text to lowercase.

Filter Words

- Remove Empty Strings:
  ➔ Filter out any empty strings.
- Remove Stop Words:
  ➔ Filter out words that are in the stop words list.
- Ensure Alphabetic Words:
  ➔ Filter to keep only alphabetic words.

```
JavaRDD<String> words = cleanedText
        .filter(s -> !s.isEmpty())
        .filter(s -> !stopWordList.contains(s))
        .filter(s -> s.matches( regex: "[a-z]+"));

words = words.map(s -> s.replaceAll( regex: "(&lt;)|>|[^\\w]+", replacement: ""));
```

Figure 15: Compare text with stop words.

Additional Cleanup (if necessary)

- Remove Special Characters:
  ➔ Further, clean up words by removing specific special characters.

```
words = words.map(s -> s.replaceAll( regex: "(&lt;)|>|[^\\w]+", replacement: ""));
```

Figure 16: Remove special characters.

Count Word Frequencies

- Map each word to a key-value pair where the key is the word and the value is 1.
- Aggregate the counts for each word using reduceByKey.

```
JavaPairRDD<String, Integer> wordCounts = words.mapToPair(word -> new Tuple2<>(word, 1))
        .reduceByKey((a, b) -> a + b);
```

Figure 17: Create pair for each word and set count default count to 1.

Output Word Frequencies:

- Collect and print the word counts in the format (word, count).

```
wordCounts.collect().forEach(
        wordCount -> System.out.println("(" + wordCount._1() + ", " + wordCount._2() + ")"));
```

Figure 18: Print each word on console.

Find and Print Maximum and Minimum Frequency Words.

- Find Maximum Frequency Word:
  - ➔ Swap the key-value pairs to facilitate sorting by frequency.
  - ➔ Sort by frequency in descending order and take the top result.
- Find Minimum Frequency Word:
  - ➔ Swap the key-value pairs to facilitate sorting by frequency.
  - ➔ Sort by frequency in ascending order and take the top result.
- Print the results for both maximum and minimum frequency words in the format (word, frequency).

```
Tuple2<Integer, String> maxFreqWord = wordCounts.mapToPair(Tuple2::swap) JavaPairRDD<Integer, String>
        .sortByKey( ascending: false)
        .take( num: 1) List<Tuple2<...>>
        .get(0);

Tuple2<Integer, String> minFreqWord = wordCounts.mapToPair(Tuple2::swap) JavaPairRDD<Integer, String>
        .sortByKey( ascending: true)
        .take( num: 1) List<Tuple2<...>>
        .get(0);

System.out.println(
        "Word with maximum frequency :  (" + maxFreqWord._2() + " ," + maxFreqWord._1() + ")");

System.out.println(
        "Word with minimum frequency : (" + minFreqWord._2() + " ," + minFreqWord._1() + ")");
```

Figure 19: Calculate the Max. and Min. Frequency of words.

Stop Spark Session

- Close the SparkSession.

```
sparkSession.stop();
```

Figure 20: Close spark session.

Program

- I created the cluster using the same method and configuration as I learned and followed in Lab 6 - BIG DATA: HADOOP AND APACHE SPARK, and I didn't put any screenshots for the cluster creation as the assignment didn't require that.
- I create a cluster with the name "spark-freqcount".
- Upload .jar, StopWords.txt, and reut2-014.sgm files to cluster.
- Here I show that the files are successfully uploaded and we can see them on the cluster using the "ls" command.

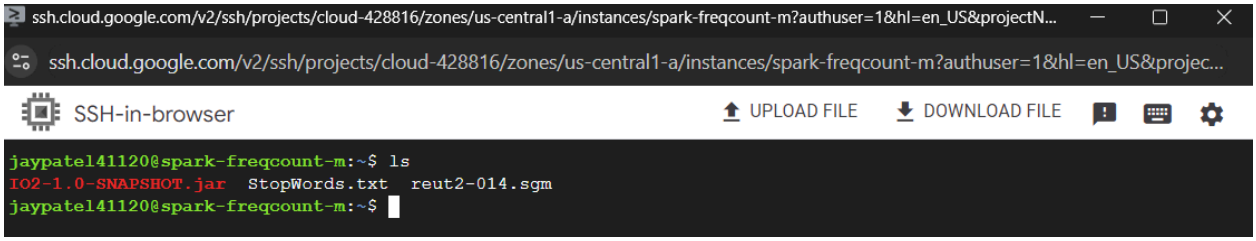

Figure 21: Upload files on cluster.

- Now I run the program using the command "spark-submit –class Program.FreqCount IO2-1.0-SNAPSHOT.jar".

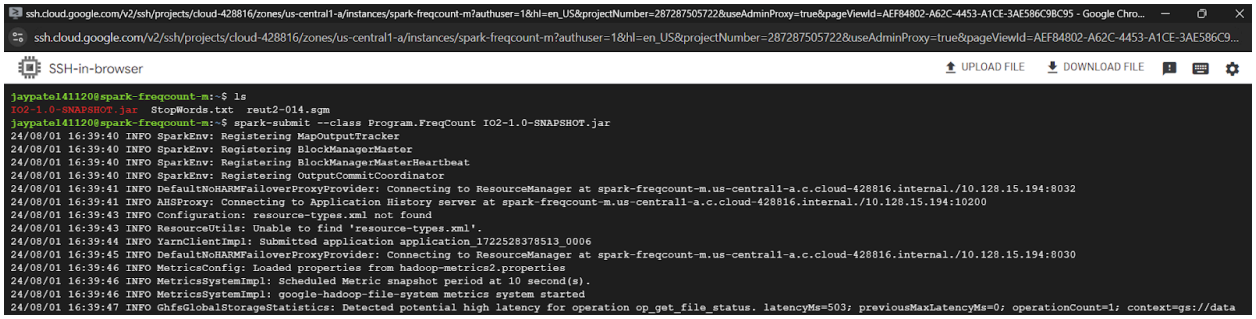

Figure 22:Run java program on cluster.

- Here in the code, I print the content of "reut2-014.sgm", and "StopWords.txt", and also display words with the highest and lowest frequency.
- Here I take all the stopwords from the GitHub repo sebleier/NLTK's list of english stopwords.

```
Content of File:
+--------------------+
|               value|
+--------------------+
|<!DOCTYPE lewis S...|
|<REUTERS TOPICS="...|
|<DATE> 7-APR-1987...|
|    <TOPICS></TOPICS>|
|    <PLACES></PLACES>|
|    <PEOPLE></PEOPLE>|
|        <ORGS></ORGS>|
|<EXCHANGES></EXCH...|
|<COMPANIES></COMP...|
|           <UNKNOWN> |
|        &#5;&#5;&#5;E|
|&#22;&#22;&#1;f11...|
|b f BC-JOHANNESBU...|
|           <TEXT>&#2;|
|<TITLE>JOHANNESBU...|
|<DATELINE>    JOH...|
|slightly firmer i...|
|reaction to a ret...|
|dlrs and a firmer...|
|    Heavyweight V...|
+--------------------+
only showing top 20 rows
```

Figure 23: Print content of file on console.

```
Stop Words:
+----------+
|     value|
+----------+
|         i|
|        me|
|        my|
|    myself|
|        we|
|       our|
|      ours|
| ourselves|
|       you|
|      your|
|     yours|
|  yourself|
|yourselves|
|        he|
|       him|
|       his|
|   himself|
|       she|
|       her|
|      hers|
+----------+
only showing top 20 rows
```

Figure 24: Print stop words on console.

```
(francs, 25)
(bone, 2)
(constraints, 3)
(drummond, 1)
(pentoxide, 1)
(yun, 1)
(brewery, 2)
(secure, 4)
(ogren, 4)
(dayton, 7)
(fred, 1)
(preventing, 1)
(end, 58)
(leeway, 2)
(barring, 1)
(financials, 1)
(sumatran, 2)
(tough, 7)
(pig, 1)
(specia, 1)
(clients, 2)
(recurrence, 1)
(koido, 2)
(gnma, 2)
(less, 24)
(supporting, 5)
(mch, 15)
(paso, 1)
(supporter, 1)
(irresponsible, 1)
(adequately, 1)
(smooth, 1)
(plus, 19)
(grant, 5)
(different, 2)
(shape, 1)
(sarasota, 1)
(completing, 2)
(phillips, 2)
(jeddah, 1)
```

Figure 25: Show random words and its frequency.

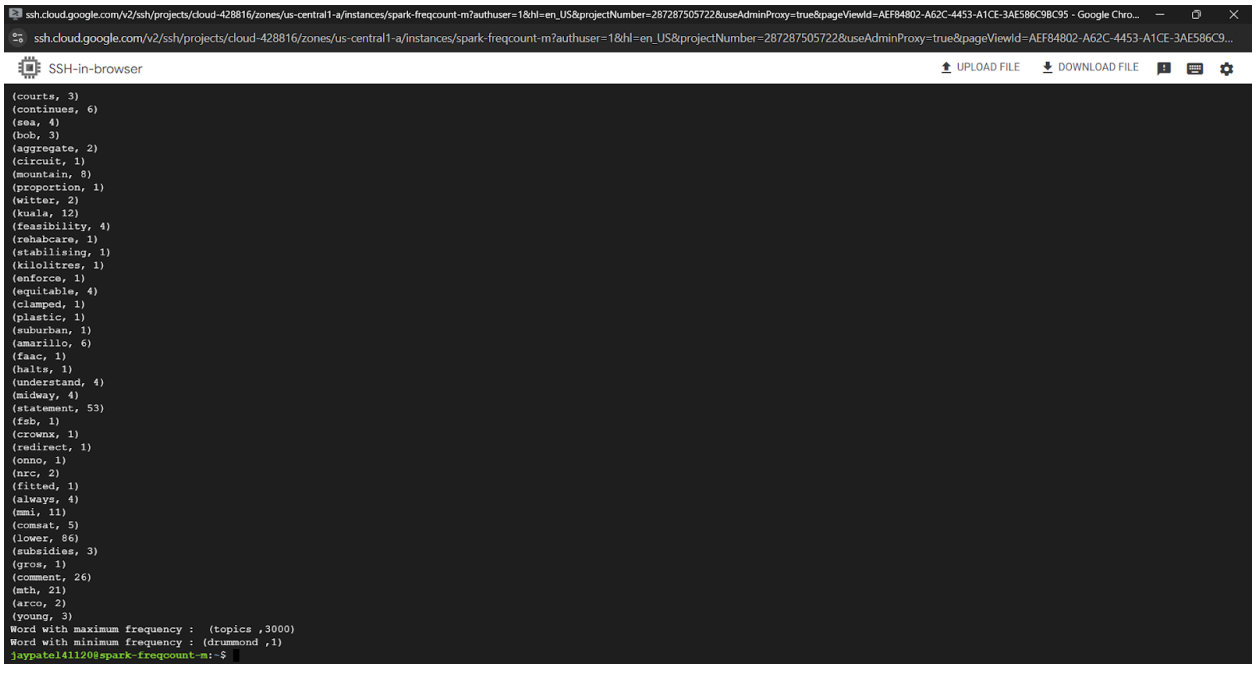

Figure 26: Show last lines of output

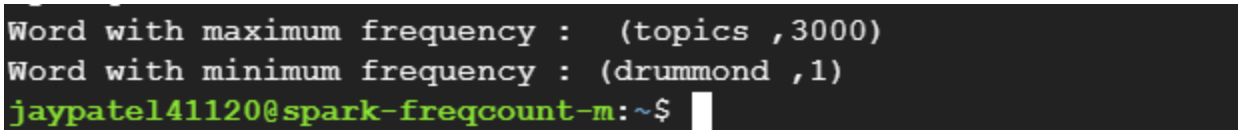

Figure 27:Show Max. and Min. Frequency.

- The word with the Max. frequency is “topics” with a frequency of 3000.
- The word with the Min. frequency is “drummond” with a frequency of 1.

Problem 2

Entities

- From Assignment 1, I have four entities as follows:

Resources

- Resource_ID
- Name
- Type

Park

- Park_ID
- Name
- Location
- Area
- Type
- Opening Hours

Site

- Site_ID
- Site Number
- Location
- Capacity
- Rate
- Status

Reservation

- Reservation_ID
- Date
- Time
- Status
- Payment Status

The relationships are:

- Resources have Parks.
- Parks have Sites.
- Sites are reserved by Reservations.

Insert Data

- ➔ Now first I insert data for all the four entities and than establish appropriate relationships.
- ➔ So here I insert data for resource entity.
- ➔ The Cypher query creates a node labeled Resource with properties Resource_ID set to 1, Name set to 'Resource1', and Type set to 'Park'.

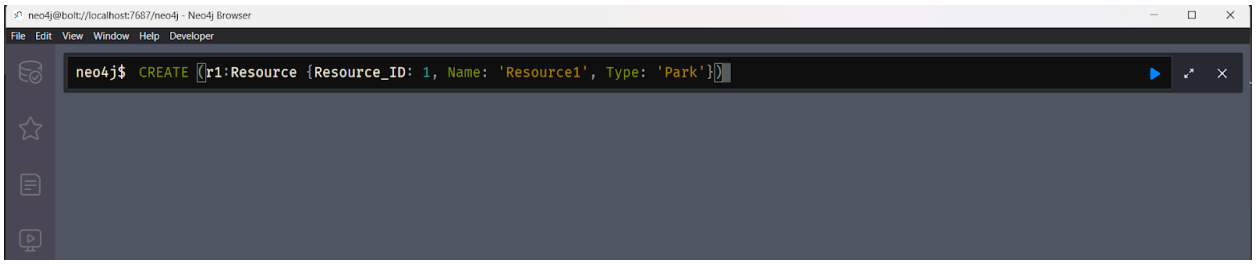

Figure 28: Insert data into resource entity.

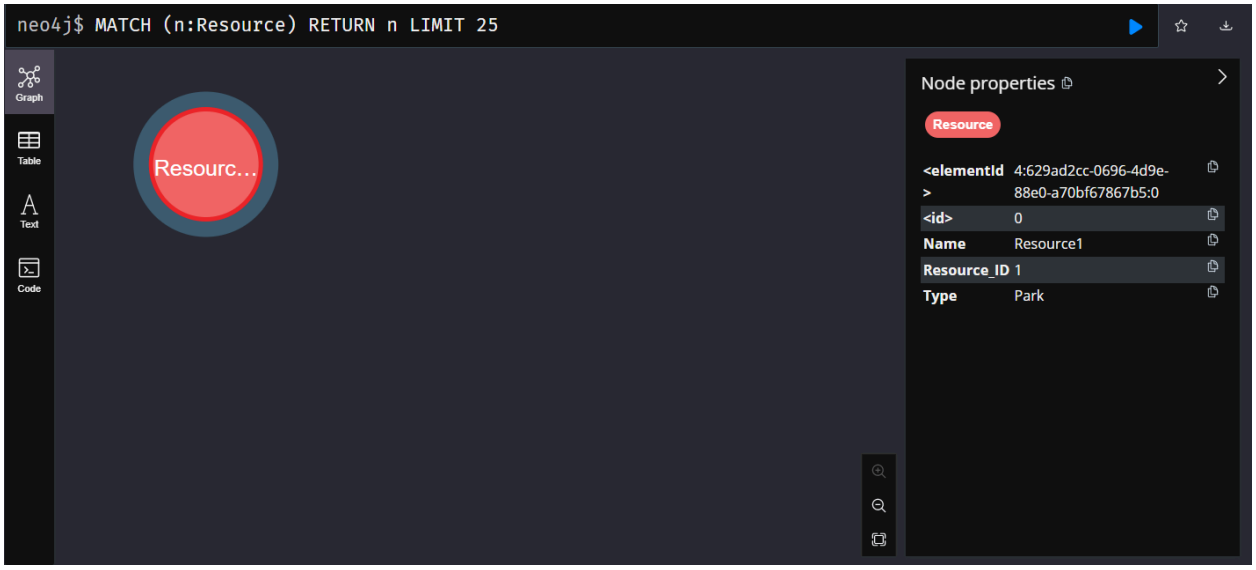

Figure 29: Show resource entity.

- ➔ Insert data for Park entity.
- ➔ The Cypher query creates a node labeled `Park` with properties: `Park_ID` set to 1, `Type` set to 'National Park', `Location` set to 'Location1', `Area` set to '5000 sq km', `Name` set to 'Park1', and `Opening_Hours` set to '9 AM - 5 PM'.

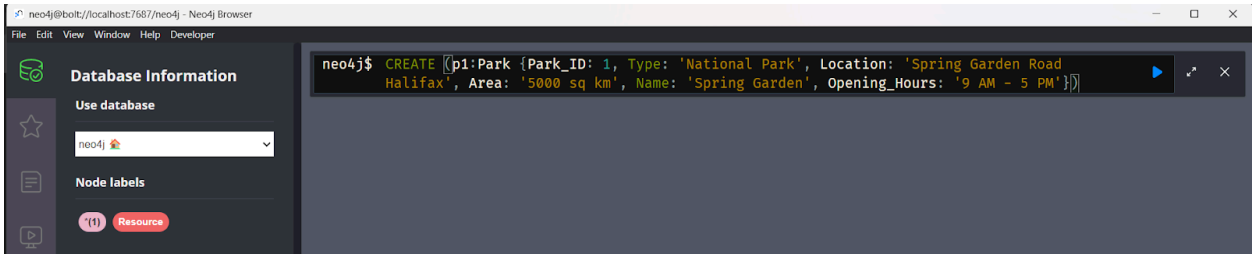

Figure 30: Insert data into park entity.

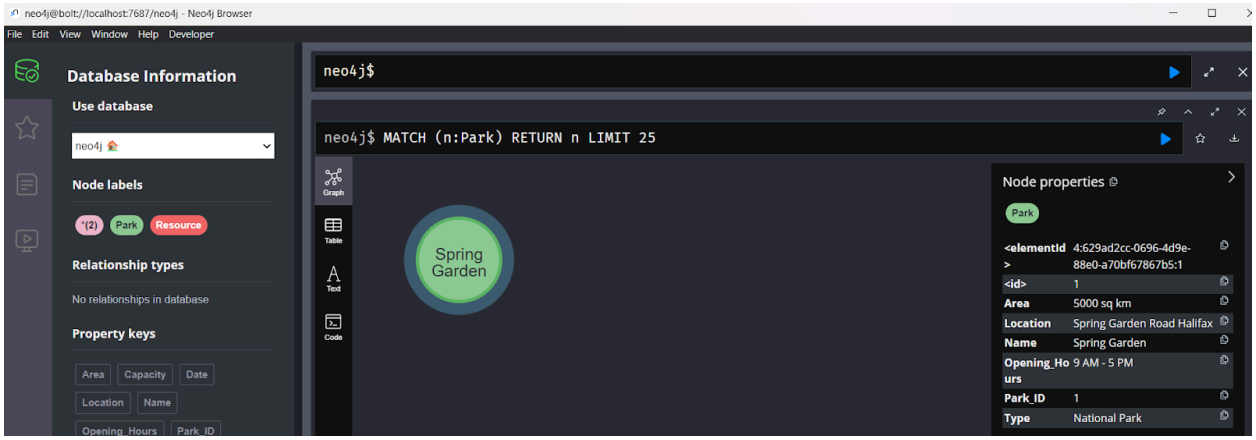

Figure 31: Show park entity.

- ➔ Insert data for Site entity.
- ➔ The Cypher query creates a node labeled `Site` with properties: `Site_ID` set to 1, `Site_Number` set to 101, `Location` set to 'LocationA', `Capacity` set to 50, `Rate` set to 100, and `Status` set to 'Available'.

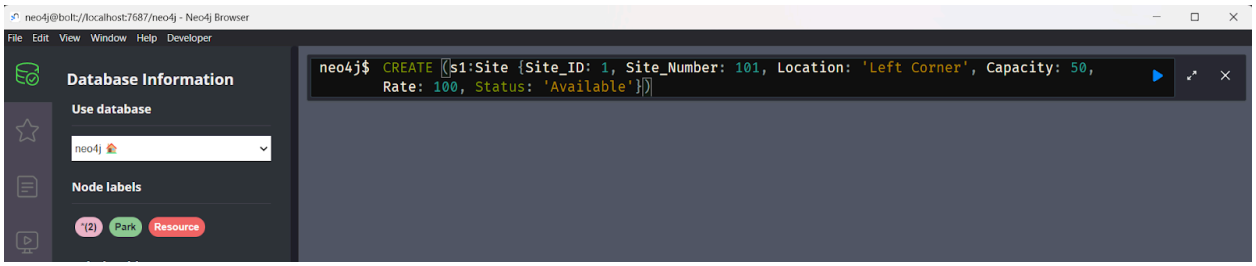

Figure 32: Insert data into site entity.

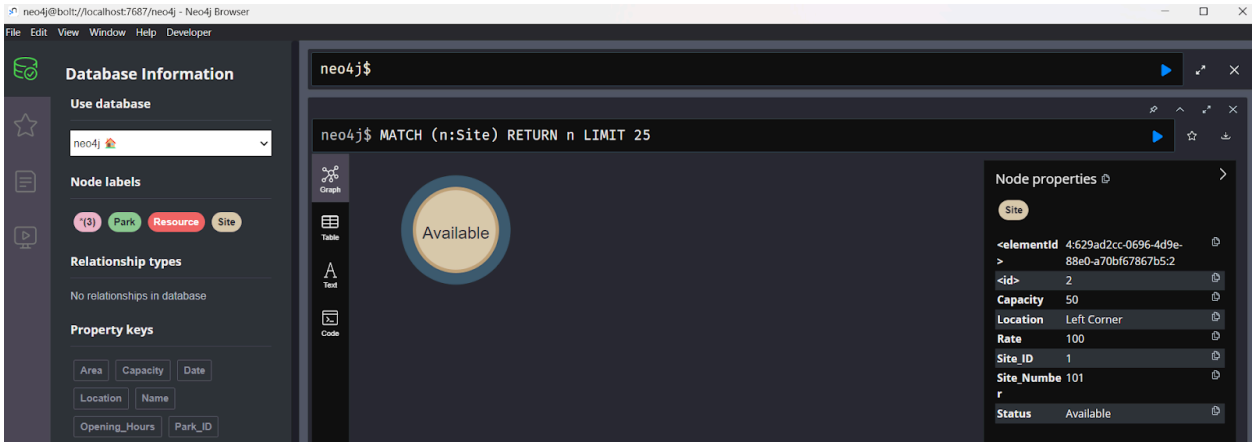

Figure 33: Show site entity.

- ➔ Insert data into Reservation entity.
- ➔ The Cypher query creates a node labeled `Reservation` with properties: `Reservation_ID` set to 1, `Date` set to '2024-08-01', `Time` set to '10:00', `Status` set to 'Confirmed', and `Payment_Status` set to 'Paid'.

Figure 34: Insert into reservation entity.

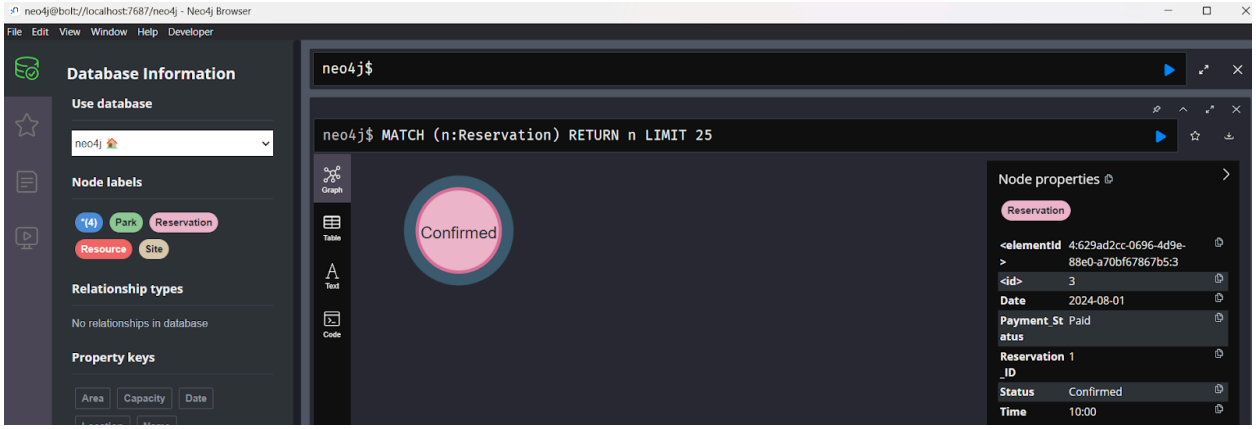

Figure 35: Show reservation entity.

Relationships

- ➔ Establish relationship between resource and park.
- ➔ So resources is handle by nova scotia.
- ➔ Resource entity has so many type like park, lake, and natural resources.
- ➔ Here we only consider park.

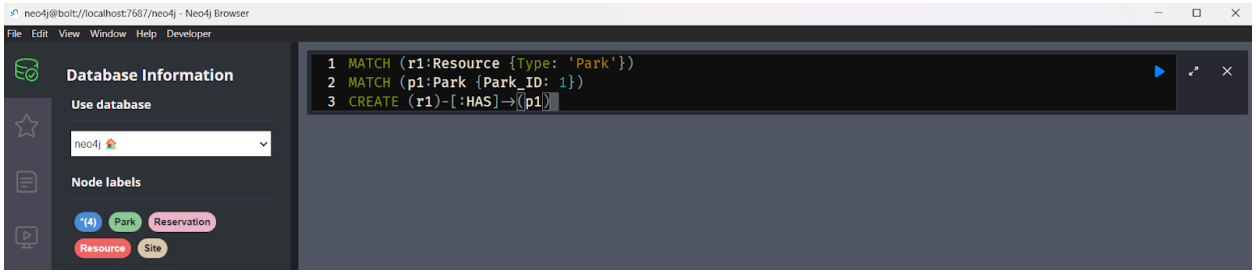

Figure 36: Establish relationship between resource and park.

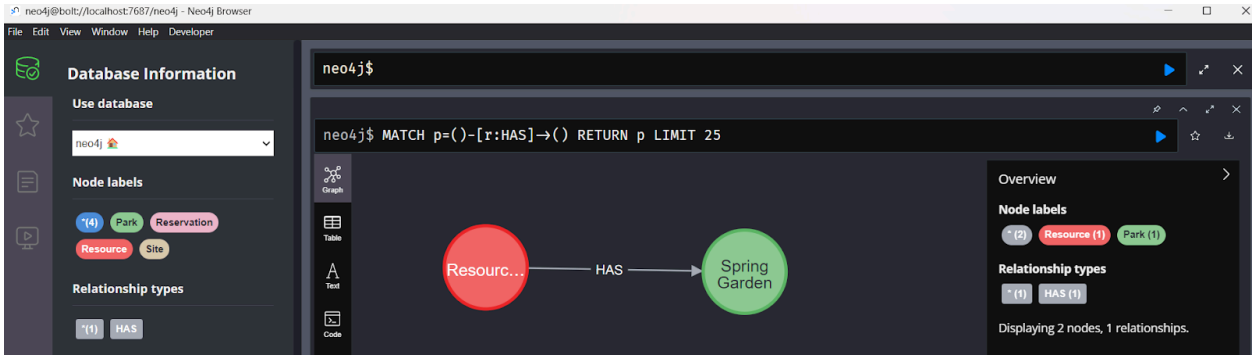

Figure 37: Show relationship between resource and park.

- ➔ Establish relationship between park and site.
- ➔ One park has many site on different locations.

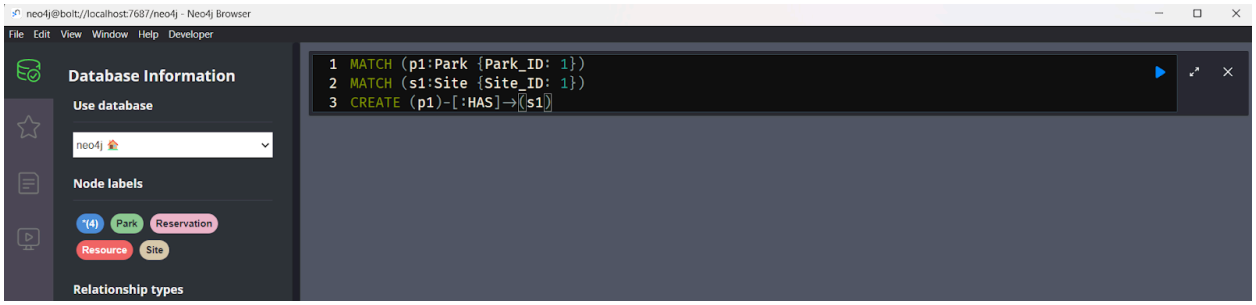

Figure 38: Establish relationship between park and site.

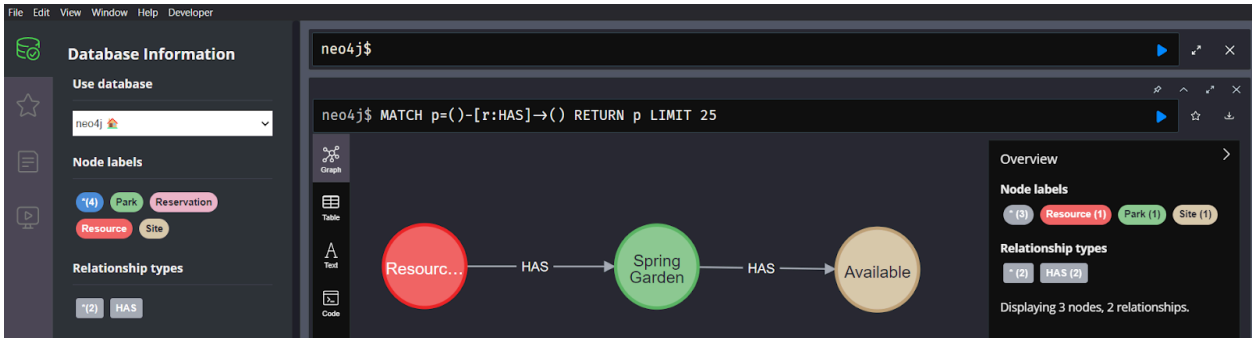

Figure 39: Show relationship between park and site.

- ➔ Establish relationship between site and reservation
- ➔ One site can be reserved by many people, but only one at a time.

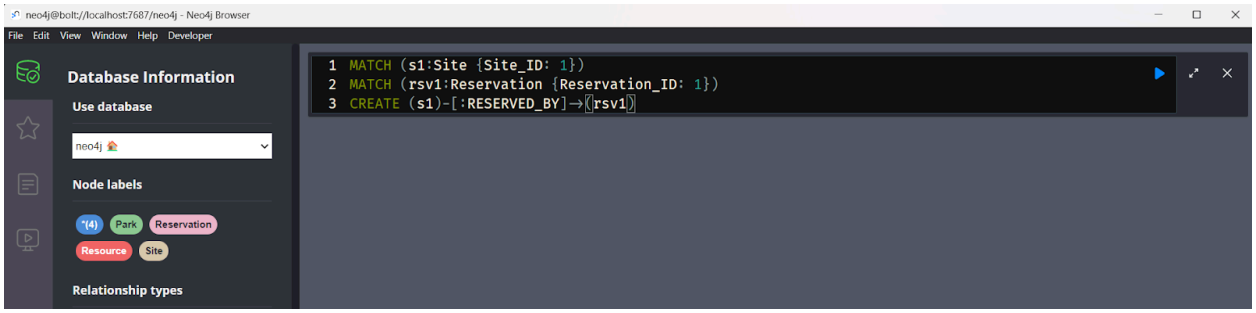

Figure 40: Establish relationship between site and reservation.

Figure 41: Show relationship between site and reservation.

Final Output Graph

- ➔ Here is the entire graph that we get from all the data we inserted and relationships we established.
- ➔ It is clearly show in graph that all the four entities are related to each other with proper relationships.

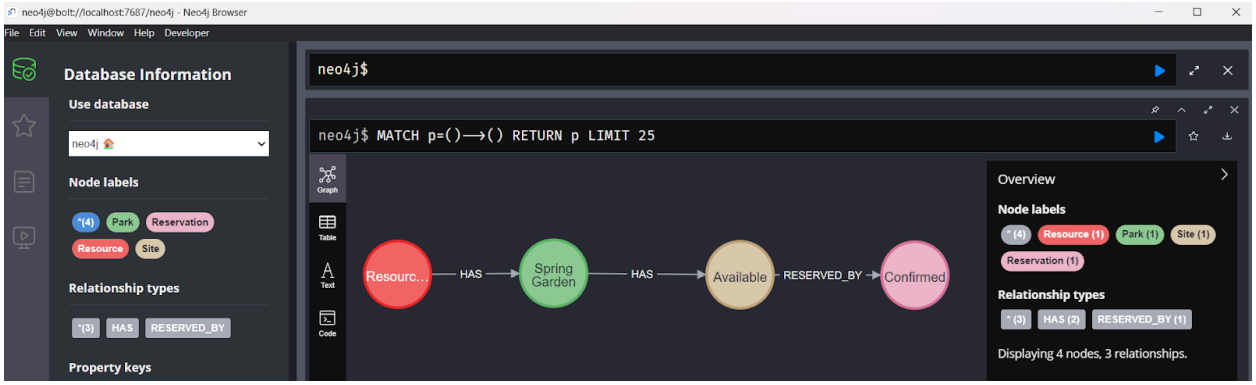

Figure 42: Show overall graph after final step.

Problem 3

Algorithm

Initialize MongoDB Connection

- Create a MongoClient and connect to MongoDB using a predefined URI.

Read and Process Data

- Read BOW Data:
  - ➔ Use FileOperations to read data from the file reut2-014.sgm.
  - ➔ Extract relevant information and filter it to get a list of string arrays.

```
1 usage
private StringBuilder readFile(String filepath){
    StringBuilder content = new StringBuilder();
    try (BufferedReader br = new BufferedReader(new FileReader(filepath))) {
        String line;
        while ((line = br.readLine()) != null) {
            content.append(line);
        }
        return content;
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
    return null;
}

1 usage
private List<String> getInfoFromRawData(StringBuilder content) {
    List<String> results = new ArrayList<>();
    String regex = "<REUTERS.*?>(.*?)</REUTERS>";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(content);
    while (matcher.find()) {
        results.add(matcher.group(1));
    }
    return results;
}
```

Figure 43: Fetch data from "reut2-014.sgm" and read data inbetween "REUTERS" tag.

```
1 usage
private List<String[]> filterData(List<String> info){
    List<String[]> response = new ArrayList<>();

    for (String currInfo : info) {
        String title = getTitleAndBody(currInfo, tagName: "TITLE");
        String filteredData = cleanTitleAndBody(title);
        if (filteredData != null){
            String[] words = filteredData.split( regex: " ");
            response.add(words);
        }
    }

    return response;
}

1 usage
private String getTitleAndBody(String content, String tagName) {
    String regex = "<" + tagName + ".*?>(.*?)</" + tagName + ">";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(content);
    if (matcher.find()) {
        return matcher.group(1);
    }
    return null;
}
```

Figure 44: Extract title from news.

```
1 usage
private String cleanTitleAndBody(String content) {
    if(content == null)
        return content;
    return content.replaceAll( regex: "[^a-zA-Z0-9\\s]", replacement: "")
            .replace( target: "&lt;", replacement: "").replace( target: ">", replacement: "")
            .replaceAll( regex: "\\s+", replacement: " ").trim();
}
```

Figure 45: Remove special characters and unnecessary spaces.

- Read Words:
  - ➔ Retrieve lists of negative and positive words from negativeWords.txt and positiveWords.txt.

```
2 usages
public List<String> getWords(String filepath){
    List<String> words = new ArrayList<>();
     try (BufferedReader br = new BufferedReader(new FileReader(filepath))) {
         String line;
         while ((line = br.readLine()) != null) {
             words.add(line);
         }
         return words;
     } catch (IOException e) {
         System.out.println(e.getMessage());
     }
     return null;
}
```

Figure 46: Fetch positiveWords and negativeWords from files.

Calculate Sentiments

- For each data entry:
    - ➔ Count occurrences of positive and negative words.
    - ➔ Determine sentiment polarity (positive, negative, or neutral).
    - ➔ Create a TitleModel object with the sentiment details.

```
1 usage
public List<TitleModel> calculateSentiments(List<String[]> data, List<String> negativeWords, List<String> positiveWords) {
    List<TitleModel> response = new ArrayList<>();
    int index = 0;

    for (String[] curr : data) {
        int positive = 0;
        int negative = 0;
        List<String> curPositiveWords = new ArrayList<>();
        List<String> curNegativeWords = new ArrayList<>();

        index++;

        for (String str : curr) {
            if (positiveWords.contains(str.trim().toLowerCase())) {
                positive++;
                curPositiveWords.add(str);
            } else if (negativeWords.contains(str.trim().toLowerCase())) {
                negative++;
                curNegativeWords.add(str);
            }
        }

        TitleModel titleModel = createModel(index, curr, positive, negative, curPositiveWords, curNegativeWords);

        response.add(titleModel);
    }

    return response;
}
```

Figure 47: Iterate through each new and calculate its sentiment.

```
1 usage
private static String convertArrayToString(String[] titleArray) {
    StringBuilder stringBuilder = new StringBuilder();
    for (String str : titleArray) {
        stringBuilder.append(str);
    }
    return stringBuilder.toString();
}

1 usage
private static Type findPolarity(int positive, int negative) {
    if (positive > negative) {
        return Type.POSITIVE;
    } else if (negative > positive) {
        return Type.NEGATIVE;
    } else {
        return Type.NEUTRAL;
    }
}
```

Figure 48: Convert title array to string and find its polarity.

```
1 usage
private TitleModel createModel(int index, String[] title, int positive, int negative, List<String> curPositiveWords, List<String> curNegativeWords){
    TitleModel titleModel = new TitleModel();
    titleModel.setId(index);
    titleModel.setTitle(convertArrayToString(title));
    titleModel.setNegativeMatch(curNegativeWords);
    titleModel.setPositiveMatch(curPositiveWords);

    Type polarity = findPolarity(positive, negative);

    if (polarity.equals(Type.POSITIVE)) {
        titleModel.setPolarity(Type.POSITIVE);
        titleModel.setMatch(curPositiveWords);
        titleModel.setScore("+" + positive);
    } else if (polarity.equals(Type.NEGATIVE)) {
        titleModel.setPolarity(Type.NEGATIVE);
        titleModel.setMatch(curNegativeWords);
        titleModel.setScore("-" + negative);
    } else {
        titleModel.setPolarity(Type.NEUTRAL);
        List<String> neutralList = new ArrayList<>();
        neutralList.addAll(curNegativeWords);
        neutralList.addAll(curPositiveWords);
        titleModel.setMatch(neutralList);
        titleModel.setScore("0");
    }

    return titleModel;
}
```

Figure 49: Create news model after calculating polarity.

Generate Excel Report

- Create an Excel workbook with a sheet labeled "Sentiments".
- Populate the sheet with the sentiment data from TitleModel objects.
- Format the workbook with appropriate headers and data rows.

```
1 usage
private static Workbook createExcel(List<TitleModel> sentiments) {
    Workbook workbook = new XSSFWorkbook();
    Sheet sheet = workbook.createSheet( s: "Sentiments");

    String[] headers = {"ID", "Title", "Match", "Positive Match", "Negative Match", "Score", "Polarity"};
    Row headerRow = sheet.createRow( i: 0);
    for (int i = 0; i < headers.length; i++) {
        Cell cell = headerRow.createCell(i);
        cell.setCellValue(headers[i]);
    }

    for (TitleModel sentiment : sentiments) {
        Row row = sheet.createRow(sentiment.getId());
        row.createCell( i: 0).setCellValue(sentiment.getId());
        row.createCell( i: 1).setCellValue(sentiment.getTitle());
        row.createCell( i: 2).setCellValue(String.join( delimiter: ", ", sentiment.getMatch()));
        row.createCell( i: 3).setCellValue(String.join( delimiter: ", ", sentiment.getPositiveMatch()));
        row.createCell( i: 4).setCellValue(String.join( delimiter: ", ", sentiment.getNegativeMatch()));
        row.createCell( i: 5).setCellValue(sentiment.getScore());
        row.createCell( i: 6).setCellValue(sentiment.getPolarity().toString());
    }

    return workbook;
}
```

Figure 50: Convert list of news to one workbook.

Upload Data to MongoDB

- Connect to the SentimentAnalysis database and access the Sentiments collection.
- Convert the Excel data into MongoDB documents.
- Insert the documents into the MongoDB collection.

```
1 usage
private static void uploadExcelSheet(Workbook workbook) {
    List<Document> documents = new ArrayList<>();
    Sheet sheet = workbook.getSheetAt( i: 0);
    for (int i = 1; i <= sheet.getLastRowNum(); i++) {
        Row row = sheet.getRow(i);
        if (row != null) {
            Document doc = new Document()
                    .append("ID", (int) row.getCell( i: 0).getNumericCellValue())
                    .append("Title", row.getCell( i: 1).getStringCellValue())
                    .append("Match", row.getCell( i: 2).getStringCellValue())
                    .append("Positive Match", row.getCell( i: 3).getStringCellValue())
                    .append("Negative Match", row.getCell( i: 4).getStringCellValue())
                    .append("Score", row.getCell( i: 5).getStringCellValue())
                    .append("Polarity", row.getCell( i: 6).getStringCellValue());

            documents.add(doc);
        }
    }
    collection.insertMany(documents);
    System.out.println("Data Inserted Successfully To MongoDB Database!!");
}
```

Figure 51: Upload data to MongoDB.

MongoDB Connection

```java
public class BOW {

    2 usages
    private static MongoClient mongoClient;
    2 usages
    private static MongoCollection<Document> collection;
    2 usages
    private static Calculations calculations;

    1 usage
    private static void connectMongoDB() {
        MongoClientURI uri = new MongoClientURI("mongodb+srv://root:root@default-cluster.xn53b0c.mongodb.net/?retryWrites=true&w=majority&appName=default-cluster");
        mongoClient = new MongoClient(uri);
    }

    public static void main(String[] args) {
        FileOperations fileOperation = FileOperations.getInstance();
        List<String[]> data = fileOperation.getBOW();
        List<String> negativeWords = fileOperation.getWords( filepath: "negativeWords.txt");
        List<String> positiveWords = fileOperation.getWords( filepath: "positiveWords.txt");
        calculations = new Calculations();

        List<TitleModel> response = calculations.calculateSentiments(data, negativeWords, positiveWords);
        Workbook workbook = createExcel(response);

        connectMongoDB();
        MongoDatabase database = mongoClient.getDatabase( databaseName: "SentimentAnalysis");
        collection = database.getCollection( s: "Sentiments");
        uploadExcelSheet(workbook);
    }
```

Figure 52: Connect to MongoDB database.

Output

- Here I upload data to MongoDB database instead of create excel file and store in local files.
- I create database name “SentimentAnalysis” and collection name “Sentiments”.

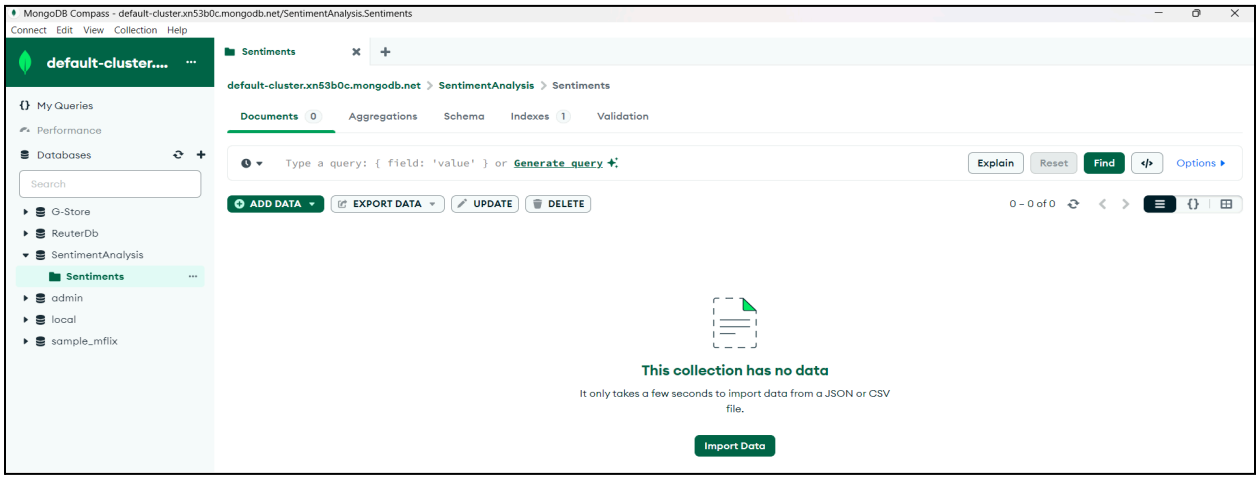

Figure 53: MongoDB database before I run Java progarm.

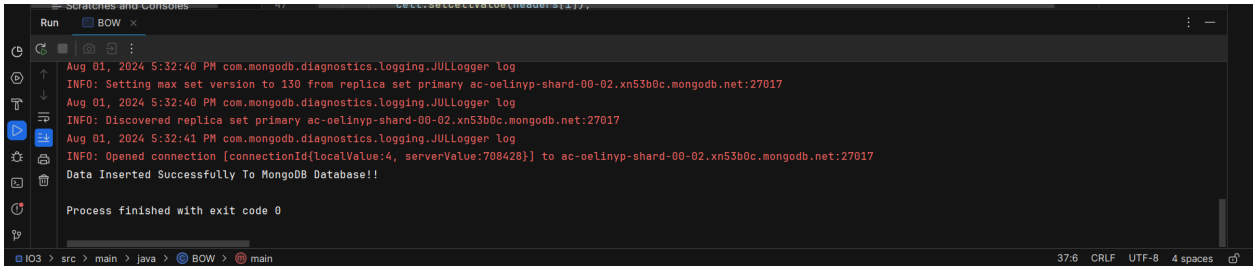

Figure 54: Run Java Program.

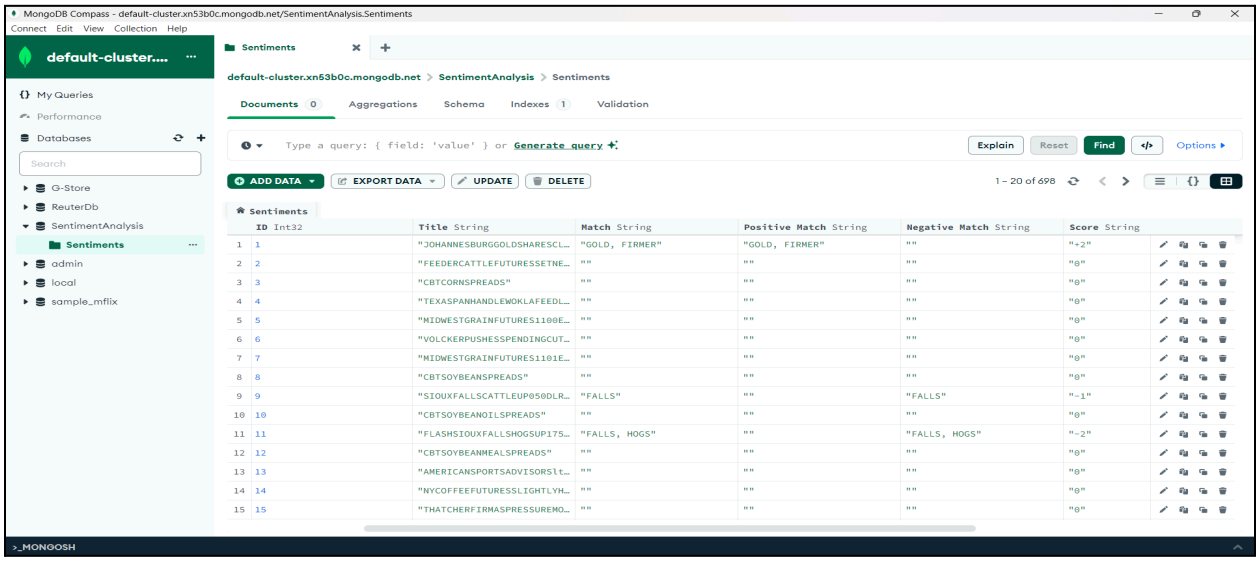

Figure 55: Data On MondoDB (Show ID, Title, Match, Positive Match, Negative Match, Score).

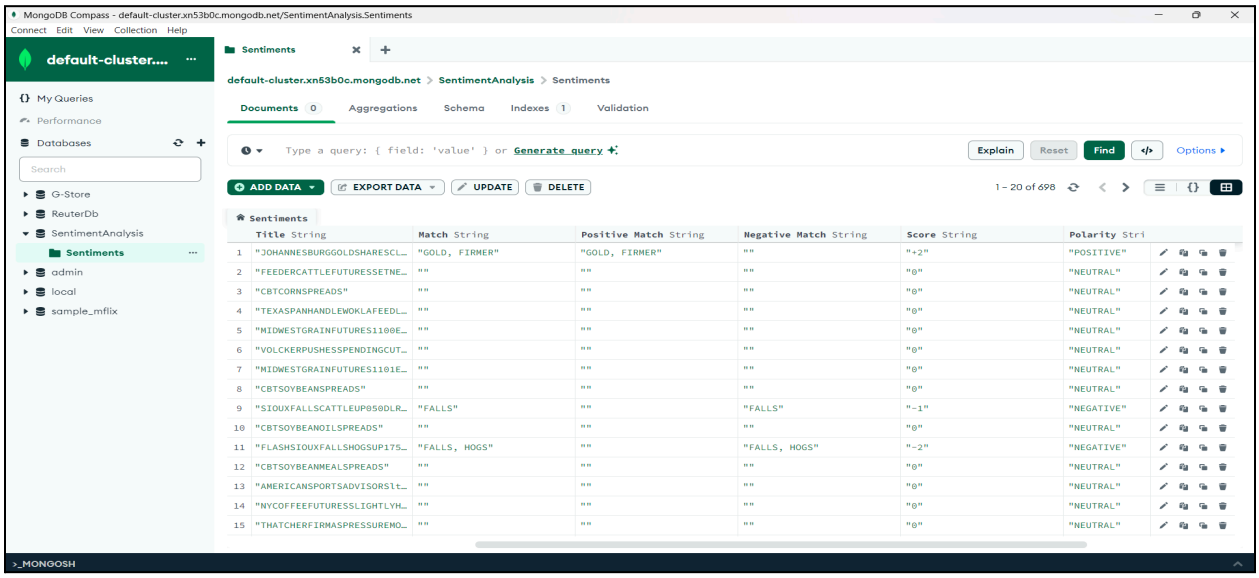

Figure 56: Data On MondoDB (Polarity).

- Here I create the excel sheet and than upload data to mongoDB, I understand that if we have to upload data than no need to cerate data in form of excel sheet we can directly upload data to mongoDB in form documents.
- I create workbook because if I didn't want to upload data on mongoDB than I just have to just write few lines of code that is FileOutputStream to store excel sheet in local files.

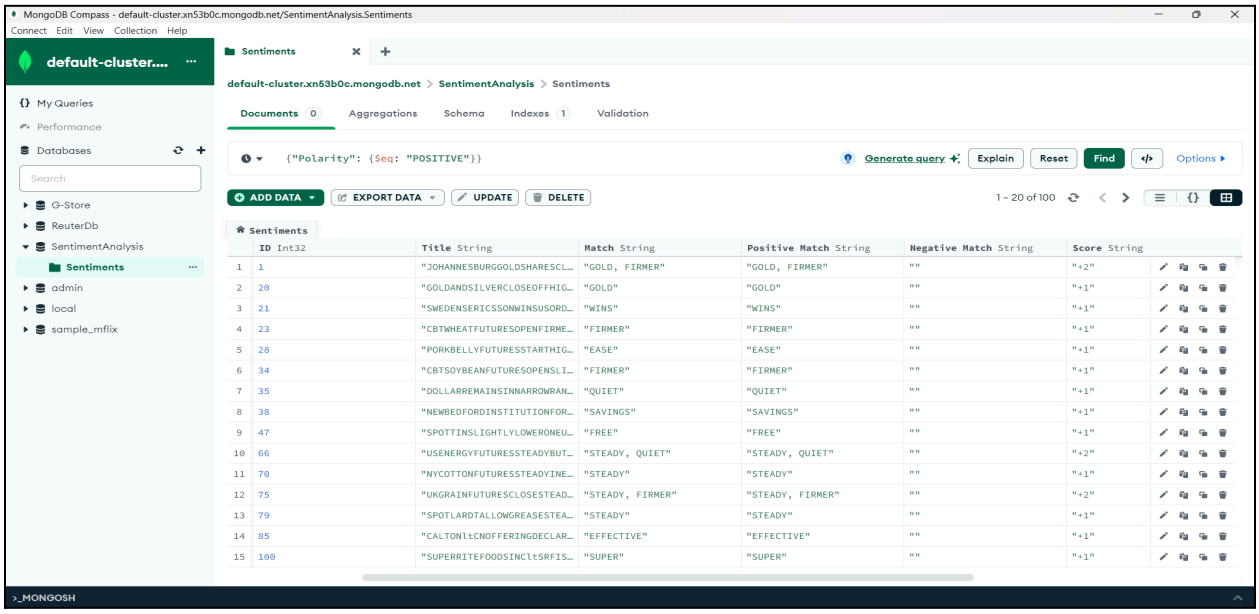

Figure 57: Show only positive sentiments news.

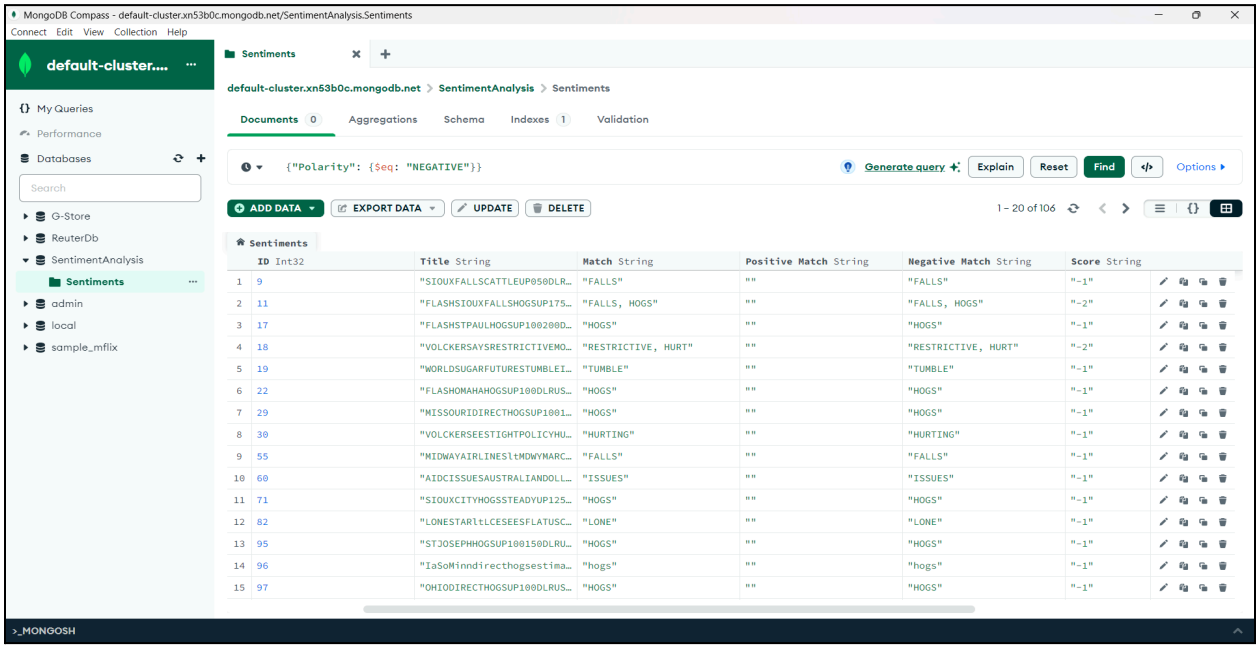

Figure 58: Show only negative sentiments news.

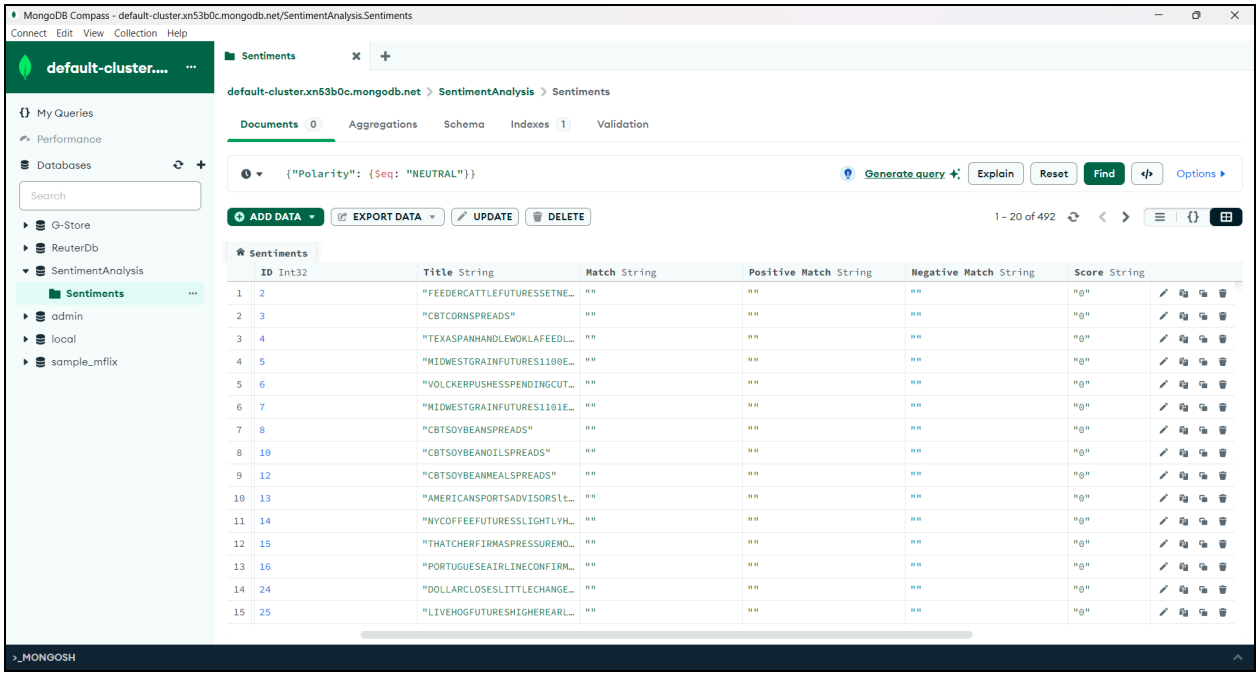

Figure 59: Show only neutral sentiments news.

- To upload data to MongoDB, I chose the following fields:
  - ID
  - Title
  - Match
  - Positive Match
  - Negative Match
  - Score
  - Polarity

- After analyzing the sentiments for all news articles, I found that:
  - A total of 698 news articles are present.
  - 100 news articles have positive sentiments.
  - 106 news articles have negative sentiments.
  - 492 news articles have neutral sentiments.

References

[1] diagrams.net, "Free Online Diagram Editor", https://app.diagrams.net/, [Accessed July 31, 2024].

[2] S. Bleier, "Stop Words List", https://gist.github.com/sebleier/554280, [Accessed July 31, 2024].

[3] Neo4j, "Cypher Query Language, "https://neo4j.com/docs/cyphermanual/current/introduction/, [Accessed July 31, 2024].

[4] M. Kulakowski, "Negative Words List", https://gist.github.com/mkulakowski2/4289441, [Accessed July 31, 2024].

[5] M. Kulakowski, "Positive Words List", https://gist.github.com/mkulakowski2/4289437, [Accessed July 31, 2024]