

**CSCI 5408**

**Data Management, Warehousing and Analytics**

**Assignment 3**

**Name : Sagarkumar Pankajbhai Vaghasia**

**CSID : vaghasia**

**Banner ID : #B00878629**

GitLab Link :

<https://git.cs.dal.ca/vaghasia/csci5408_s22_sagarkumar_vaghasia_b00878629/-/tree/main/A3>

**Problem #1**

**Task – 1 : News Article Data Extraction and Transformation**

I have created three classes named as Main, NewsExtraction and NewsFiltration.

The NewsExtraction class is responsible for extracting news data from the website <https://newsapi.org/> [1] and then I stored the extracted news data unfiltered in text files. I have code in such a way that each text file should contain three articles. I have searched the articles based on the keywords Canada, University, Dalhousie University, Halifax, Canada Education, Moncton, Toronto, Oil, and Inflation.

For each keyword, I have permitted to extract hundred articles. The news articles are stored in TXT files. I have named created a directory named “newsdata” and stored all the articles file in that file.

The naming convention for the text files I used is <search\_keyword\_lowercase>\_<current\_time>.txt. To exemplify, canada\_1657567359840.txt and dalhousieuniversity\_1657640150597.txt

Also, I have used JSON-java library [2] in the NewsExtraction to store the response from NewsAPI.

**Main.java**

This class do not have any methods. It is used to call the methods of NewsExtraction and NewsFiltration by using objects.

**NewsExtraction.java**

I have created static variables to store directory name, newsapi url, newsapi key, articles permitted, category keywords.

Methods in NewsExtraction :

* **extractNewsData()**

This method acts like a controller which I used to call different methods in the class.

I have used for loop for all the keywords to fetch and store the news articles by calling relevant methods.

* **fetchNews()**

This method is responsible to fetch news articles from newsapi website where I passed the argument as category keyword. I have used HttpRequest, HttpClient and HttpResponse. I have stored the fetched data in String variable.

* **prepareAndStoreNews()**

This function is responsible prepare and store news articles. Here, I have passed the arguments category keyword and the obtained news article string.

* **storeNews()**

This method is responsible to store news in the local disk. If the directory is not there to store files, then this function will create new directory.

The below attached figure displays all the text files containing news articles extracted for search keywords.

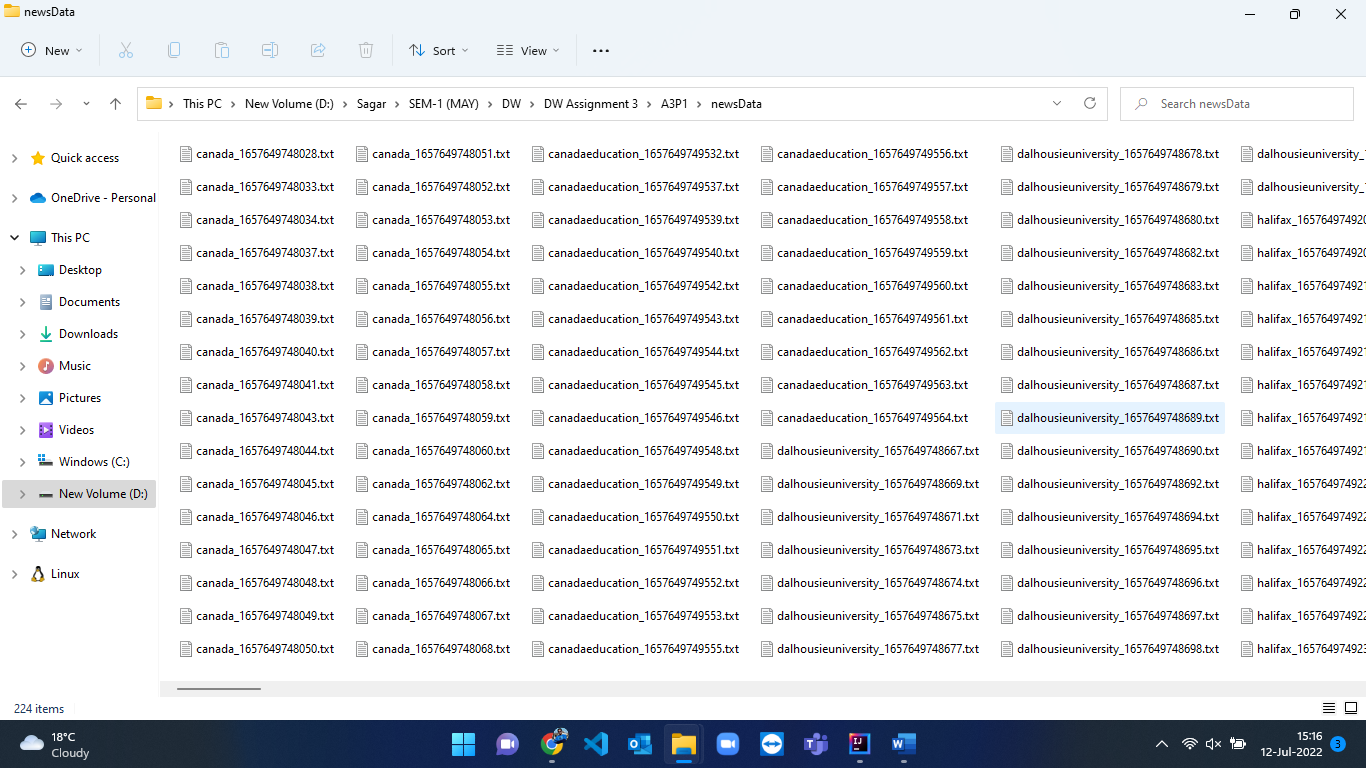


Figure 1 - Text files after extraction of news articles.

**NewsFiltration.java**

This java class I have created to filter the news articles that is stored in the directory and to upload the filtered data to the mongodb database [3].

Firstly, I have declared static variables such as SEPARATOR\_REPLACEMENT, SEPARATOR\_SPLIT, EMPTY\_STRING for filtration. Then, I have made connection with the mongodb atlas by using the url for connection.

Methods in NewsFiltration :

* **filterNewsData()**

This method acts like a controller which I used to call different methods in the class.Firstly, I read all the files that are stored in a directory in the form of array. Then, I called readAndFilterArticles method to filter the articles and then I have inserted the obtained filtered data to the mongodb database. I have also created arraylist for storing the mongoNewsDocuments in the form of list.

* **readAllFileNames()**

This method is responsible for reading the files that are stored in the directory newsdata.

* **readAndFilterArticles()**

In this method, I have passed two arguments : allNewsFiles and mongoNewsDocuments. This function is responsible for calling filterArticleContent() method and store the obtained filtered content in the String filteredArticleList. In this method, I have also parsed the obtained filtered articles data and added it to the mongoNewsDocuments.

* **filterArticleContent()**

This method contains one argument of String for the content of the article (articleContent). This method is responsible for actual filtration where I have used regular expressions [4][5][6][7] to filter the articles.

In this method, I have replaced emojis, urlToImage, url, author, id, and HTML tags from all articles with empty string.

After filtering, the obtained String is parsed into Mongodb document and stored as a list. At the end, after filtering all the articles a connection is made with Mongodb by using a connection string. Then, using insertMany() command, I have uploaded the documents into the myMongoNews database.

**Algorithm for NewsFiltration**

1. Start
2. Read name of all the text file that is extracted.
3. Initialize an empty list of Mongdb documents (mongoNewsDocuments).
4. Initialize counter variable to count the total number of articles read totalArticlesRead.
5. Iterate through all the text news articles file names.
6. All files reading finish? If yes, then go to step 11 else go to step 7
7. Read content of the current news article file.
8. Filter the content of the current news article file using the custom regexes.
9. Split the filtered news articles file content to separate news articles.
10. Parse each news article as a Mongodb document and store it in a list of Mongodb document and increment totalArticlesRead
11. Display total articles read and filtered(totalArticlesRead).
12. Connect to Mongodb atlas using mongo connection uri
13. Get the reference to the mongo database myMongoNews
14. Insert the created list of mongo news articles documents.
15. Display article stored successfully.
16. Stop

The attached figure displays the news articles uploaded to Mongodb database myMongoNews.

Graphical user interface, text, application, email

Description automatically generated

Figure 2 – NewsData uploaded to Mongodb database myMongoNews

**Task – 2 : Movie Data Extraction and Transformation**

I have created three classes named as Main, MovieDBExtraction and MovieFiltration.

The MovieDBExtraction class is responsible for extracting movies data from the website <https://www.omdbapi.com/> [8] and then I stored the extracted movies data unfiltered in text files. I have code in such a way that each text file should contain three articles. I have searched the articles based on the keywords Canada, University, Moncton, Halifax, Toronto, Vancouver, Alberta, and Niagara.

For each keyword, I have permitted to extract hundred articles. The movie articles are stored in TXT files. I have named created a directory named “moviesdata” and stored all the articles file in that file.

The naming convention for the text files I used is <search\_keyword\_lowercase>\_<current\_time>.txt. To exemplify, university\_1657650860528.txt and niagara\_1657650862012.txt

Also, I have used JSON-java library [2] in the MovieDBExtraction to store the response from omdbapi.

**Main.java**

This class do not have any methods. It is used to call the methods of MovieDBExtraction and MovieFiltration by using objects.

**MovieDBExtraction.java**

I have created static variables to store directory name, moviesapi url, moviesapi key, articles permitted, category keywords.

Methods in MovieExtraction :

* **extractMoviesData()**

This method acts like a controller which I used to call different methods in the class.

I have used for loop for all the keywords to fetch and store the movies articles by calling relevant methods.

* **fetchMovies()**

This function is responsible to fetch movies articles from omdbapi website where I passed the argument as category keyword. I have used HttpRequest, HttpClient and HttpResponse. I have stored the fetched data in String variable.

* **prepareAndStoreMovies()**

This method is responsible prepare and store movie articles. Here, I have passed the arguments category keyword and the obtained movie article string.

* **storeMovies()**

This method is responsible to store movies in the local disk. If the directory is not there to store files, then this function will create new directory.

The below attached figure displays all the text files containing movie articles extracted for search keywords.

Graphical user interface, application

Description automatically generated

Figure 3 - Text files after extraction of movie articles.

**MoviesFiltration.java**

This java class I have created to filter the movies articles that is stored in the directory and to upload the filtered data to the mongodb database [3].

Firstly, I have declared static variables such as SEPARATOR\_REPLACEMENT, SEPARATOR\_SPLIT, EMPTY\_STRING for filtration. Then, I have made connection with the mongodb atlas by using the url for connection.

Methods in MoviesFiltration :

* **filterMovieData()**

This method acts like a controller which I used to call different methods in the class.

Firstly, I read all the files that are stored in a directory in the form of array. Then, I called readAndFilterArticles method to filter the articles and then I have inserted the obtained filtered data to the mongodb database. I have also created arraylist for storing the mongoMoviesDocuments in the form of list.

* **readAllFileNames()**

This method is responsible for reading the files that are stored in the directory moviesdata.

* **readAndFilterArticles()**

In this method, I have passed two arguments : allmoviesFiles and mongoMoviesDocuments. This function is responsible for calling filterArticleContent() method and store the obtained filtered content in the String filteredArticleList. In this method, I have also parsed the obtained filtered articles data and added it to the mongoMoviesDocuments.

* **filterArticleContent()**

This method contains one argument of String for the content of the article (articleContent). This method is responsible for actual filtration where I have used regular expressions [4][5][6][7] to filter the articles.

In this method, I have replaced emojis, urlToImage, url, author, id, and HTML tags from all articles with empty string.

After filtering, the obtained String is parsed into Mongodb document and stored as a list. At the end, after filtering all the articles a connection is made with Mongodb by using a connection string. Then, using insertMany() command, I have uploaded the documents into the myMongoMovie database.

**Algorithm for NewsFiltration**

1. Start
2. Read name of all the text file that is extracted.
3. Initialize an empty list of Mongdb documents (mongoMoviesDocuments).
4. Initialize counter variable to count the total number of articles read totalArticlesRead.
5. Iterate through all the text movie articles file names.
6. All files reading finish? If yes, then go to step 11 else go to step 7
7. Read content of the current movie article file.
8. Filter the content of the current movie article file using the custom regexes.
9. Split the filtered movie articles file content to separate movie articles.
10. Parse each movie article as a Mongodb document and store it in a list of Mongodb document and increment totalArticlesRead
11. Display total articles read and filtered(totalArticlesRead).
12. Connect to Mongodb atlas using mongo connection uri
13. Get the reference to the mongo database myMongoMovie
14. Insert the created list of mongo movie articles documents.
15. Display article stored successfully.
16. Stop

The below attached figure displays the movie articles uploaded to Mongodb database myMongoMovie.

Graphical user interface, text, application, email

Description automatically generated

Figure 4 – MovieData uploaded to Mongodb database myMongoMovie.

**Task – 3 : Reuter News Data Reading and Transformation**

I have created five classes named as Main, Reuter, ReuterParse, ReuterStore and MongoDBConnection.

**Main.java**

This class do not have any methods. It is used to call the methods of ReuterParse and ReuterStore by using objects. I have also assigned connection url for Mongodb to databaseString

**ReuterParse.java**

I have created a static string array to store file names. This class is responsible for extraction of reuter data and then filtering that extracted data.

Methods in ReuterParse :

* **parseAllFiles()**

This method iterates through all the file names and call parseSGM() to extract the reuter data from the file.

* **parseSGM()**

This method takes one parameter which is file name. This method is responsible for reading file into string by using filterReuter() [9]. After filtration, it passes the string from the regular expression to obtain the reuter data.

* **filterReuter()**

This method is responsible for filtering the reuter data. I have replaced url, special word, white space, special character and emojis by empty string.

**MongoDBConnection.java**

This class is responsible for connection with the Mongodb Atlas. The constructor of this class creates the default objects for the connection. This class also contains the objects of MongoClient, MongoDatabase and ConnectionString.

Methods in MongoDBConnection:

* **getCollection()**

In this method, I passed one parameter of collection name and returns the mongodb collection object.

* **setDatabase()**

In this method, I passed one parameter for the database name and returns the mongodb database object

**Reuter.java**

This class has two String variables: title and text. The constructor of this class sets the defines the values. This class only contains getters and setters for title and text.

Methods in Reuter: setTitle(), getTitle(), setText(), getText().

**ReuterStore.java**

This class is responsible for storing the extracted and filtered data to mongodb database. This has MongoDBConnection and MongoCollection objects.

Methods in ReuterStore:

* **storeAll()**

This method takes HashMap as a parameter and it iterates through all map entry in HashMap and get the key(file name) and value pairs(list of reuter objects). After that, it store() method.

* **store()**

This method is responsible for storing the document to the mongodb database. This method takes reuter object as a parameter. After that, it creates a document object and then it calls insertOne() method to upload that document to the database.

**Algorithm for Reuter Data Cleaning and Transformation.**

1. Start
2. Read the reuter files.
3. Filter the content by using the custom regexes.
4. Write the filtered string to the text file.
5. Parse the clean string from the regular expression to extract information between <REUTERS></REUTERS> tags.
6. The information extracted between the REUTERS tags will be parsed to extract the information between <TEXT></TEXT> tags.
7. The information extracted between the TEXT tags will be parsed to extract the information between <TITLE></ TITLE> tags.
8. The information extracted between the TITLE tags will be parsed to extract the information between <BODY></ BODY> tags.
9. Create the reuter class object for each text and title .
10. Create connection to the Mongodb database and upload the data to the Mongodb database.
11. Stop

The below attached figures displays the reuter documents uploaded to Mongodb database ReuterDb.

Graphical user interface, text, application, email

Description automatically generated

Figure 5 – Reuter Documents uploaded to Mongodb database ReuterDb.

Graphical user interface, text, application, email

Description automatically generated

Figure 6 – ReuterData uploaded to Mongodb database ReuterDb.

**Problem #2**

**Task – 1 : Data Processing Using Spark – MapReduce to perform count**

Firstly, I have created the vm instance named as vminstance1 on GCP [10].

Then, on the left side panel, below the VPC network, click Firewall option.

Edit settings for default-allow-http and default-allow-https to make it allow all.

The below attached figure displays the created virtual machine (VM) instance vminstance1.

Graphical user interface, text, application, email

Description automatically generated

Figure 7 –vminstance1 on GCP.

Then, I connect to the vminstance1 shell by clicking SSH under connect option.

After that, I downloaded the tar file of Apache spark using wget utility [11] [12]. Then, I extracted the tar file and moved extracted files to /opt/sparkfolder.

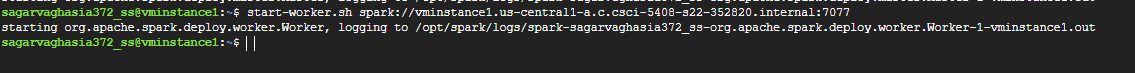
After moving to that folder, I added path of the spark folder to $SPARK\_HOME environment variable and I have also added path of bin folder inside spark folder to $PATH environment

I started master node by using the command : start-master.sh

Text

Description automatically generated

I started worker node by using the command : start-worker.sh spark://vminstance1.us-central1-a.c.csci-5408-s22-352820.internal:7077



The below attached screenshot is of master and worker node started in spark.

Graphical user interface, text, email

Description automatically generated

**MapReduce**

I have created two classes Main and WordCounterEngine.

**Main.java**

This class is used for calling methods of the WordCounterEngine and performing basic operations. It holds the object of WordCounterEngine class.

I have defined two variables maxWordCount and minWordCount and assigned -1 to both variables. I have used Map for finding the frequency of the word.

At the end, I have displayed frequency of each word alongwith word with highest and lowest frequency.

**WordCounterEngine.java**

This class is responsible for counting the frequency of searched keywords in the news files. The search keywords are Canada, University, Dalhousie University, Halifax, Canada Education, Moncton, Toronto, Oil, and Inflation.

Methods in WordCounterEngine:

* **initWordCounterMap()**

This function creates hashmap of the keywords to be searched as a string and their frequency count as a value.

* **readAllFileNames()**

This method iterates through all files and then it reads contents of each file and performs map and reduce operations.

* **map()**

This method reads the articles of news present in each file by using the Matcher and Pattern classes. It compiles the regular expressions for extracting title and content. The matched substrings for the title and content are appended to a StringBuilder object whose string value is returned.

* **reduce()**

This method is invoked for every string returned by the map() method. It will search for the keyword in the string until the last index is reached and increase the word count of the relevant keyword in the wordCounterMap hashmap.

**Algorithm for WordCounter**

1. Start
2. Initialize variables wordCounterMap and initialWordCount.
3. Initialize File array allNewsFiles to store the name of all the news articles files.
4. If wordCounterMap is empty or if allNewsFiles is null or empty, then go to step 16
5. Read the contents of the file.
6. Initialize titleMatcher and contentMatcher to match the pattern compiling a regex.
7. Initialize mappedStringBuilder to an empty StringBuilder object.
8. Append all the matches found by titleMatcher and contentMatcher to mappedStringBuilder.
9. Iterate through all the keywords.
10. Initialize the variable lastEncounterIndex to 0 for the keyword to be searched.
11. Iterate till lastEncounterIndex is not equal to -1.
12. Find the index of the occurrence of the current keyword to be searched from the lastEncounterIndex and assign it to lastEncounterIndex.
13. If the lastEncounterIndex is not equal to -1, then go to step 14.
14. Increment the count of the current keyword.
15. Continue till all files are read.
16. Stop

**Steps to submit job to spark cluster:**

* Using mvn install command in the IDE, I created the jar file.
* Then, I have uploaded the code for mapreduce, jar file and text files to git repository and cloned that git repository to the cloud instance.
* After that, I submitted my jar file to the spark master node to run using the spark-submit command [13].

spark-submit –master spark://34.135.163.120:7077 --deploy-mode cluster --class org.example.Main target/A3P3-1.0-SNAPSHOT.jar

* After executing the above command, the driv­­­­­er will be in a finished state, means that the jar file successfully run on the Spark Cluster.

Output screenshot on spark’s worker node’s log page.

Graphical user interface, text, application

Description automatically generated

**Task – 2 : Neo4j Database**

Neo4j[14] is a graph database which has nodes edges and properties. It is more suitable for certain big data and analytics applications. A graph database is used to represent relationships. It uses Cypher Query Language (CQL) [15][16].

**Queries:**

1. **Creating a node**

This query creates the Person node of name sagar

CREATE (p:Person {name: 'Sagar'}) RETURN p

Graphical user interface

Description automatically generated

1. **Displaying nodes**

This cypher query represents all the person nodes with limit of 10 results.

MATCH (p:Person) RETURN p limit 10

Graphical user interface

Description automatically generated with medium confidence

1. **Finding nodes by using MATCH**

This query will get all the movies that were released between year 2000 and 2020.

MATCH (m:Movie) where m.released > 2000 and m.released < 2020 RETURN m

A picture containing graphical user interface

Description automatically generated

1. **Relationships**

This cypher query represents all the Person nodes who directed a movie that was released after 2000. I have limited the number of result to 10.

MATCH (p:Person)-[a:ACTED\_IN]-(m:Movie) where m.released > 2000 RETURN p,a,m limit 10

Graphical user interface

Description automatically generated with medium confidence

**Summary:**

In this task of learning about Neo4j database, I explored through the graph databases which is more useful in big data and analytics applications when there is huge amount of data. It has many advantages, and it is more convenient than RDBMS.

I have walked through the basic tutorial of the Neo4j database and also learnt many queries and structures of Cypher Query Language.

I have learnt about creating node, adding properties, building relationships, deleting nodes, deleting relationships, finding nodes and many more.

I learnt that this database represents the nodes and relationships among them in the form of graph which is faster, efficient while searching.

Neo4j performs traversing related operations(such as spanning tree, DFS, BFS, etc.) very smoothly and fastly compared to other databases.

In future, if I will be working on machine learning related fields where I have to work related to recommendations or suggestions at that time I’ll be working on graph database as it is much faster and efficient compared to RDBMS.

**References :**

[1] “News API – search news and blog articles on the web,” *News API*. [Online]. Available: https://newsapi.org/. [Accessed: 08-Jul-2022].

[2] JSON-Java, *Mvnrepository.com*. [Online]. Available: https://mvnrepository.com/artifact/org.json/json/20220320. [Accessed: 08-Jul-2022].

[3] “Atlas,” *MongoDB*. [Online]. Available: https://www.mongodb.com/atlas/database. [Accessed: 08-Jul-2022].

[4] “What is the regex to extract all the emojis from a string?,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/24840667/what-is-the-regex-to-extract-all-the-emojis-from-a-string. [Accessed: 08-Jul-2022].

[5] “Regex pattern including all special characters,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/18057962/regex-pattern-including-all-special-characters/18058074. [Accessed: 08-Jul-2022].

[6] “Regular expression to remove HTML tags from a string,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/11229831/regular-expression-to-remove-html-tags-from-a-string. [Accessed: 08-Jul-2022].

[7] “Regular expression to find URLs within a string,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/6038061/regular-expression-to-find-urls-within-a-string. [Accessed: 08-Jul-2022].

[8] “OMDb API - the open movie database,” *Omdbapi.com*. [Online]. Available: https://www.omdbapi.com/. [Accessed: 08-Jul-2022].

[9] “SGML parser in java?,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/4867894/sgml-parser-in-java. [Accessed: 08-Jul-2022].

[10] “Cloud computing services,” *Google Cloud*. [Online]. Available: https://cloud.google.com/. [Accessed: 08-Jul-2022].

[11] D. Tucakov, “How to install Spark on Ubuntu,” *Knowledge Base by phoenixNAP*, 13-Apr-2020. [Online]. Available: https://phoenixnap.com/kb/install-spark-on-ubuntu. [Accessed: 08-Jul-2022].

[12] “Downloads,” *Apache.org*. [Online]. Available: https://spark.apache.org/downloads.html. [Accessed: 10-Jul-2022].

[13] “Submitting Applications - Spark 3.3.0 Documentation,” *Apache.org*. [Online]. Available: https://spark.apache.org/docs/latest/submitting-applications.html. [Accessed: 10-Jul-2022].

[14] “Graph data platform,” *Neo4j Graph Data Platform*, 16-May-2020. [Online]. Available: https://neo4j.com/. [Accessed: 11-Jul-2022].

[15] “Neo4j tutorial,” *Tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/neo4j/index.htm. [Accessed: 11-Jul-2022].

[16] “Neo4j tutorial,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/neo4j-tutorial. [Accessed: 11-Jul-2022].