DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

CSCI 5408 – Data Management, Warehousing, and Analytics

Assignment 3

Anqi Chen

B00838586

Instructor: Dr. Saurabh Dey

Date of submission: November 13, 2020

1. Cluster Setup:

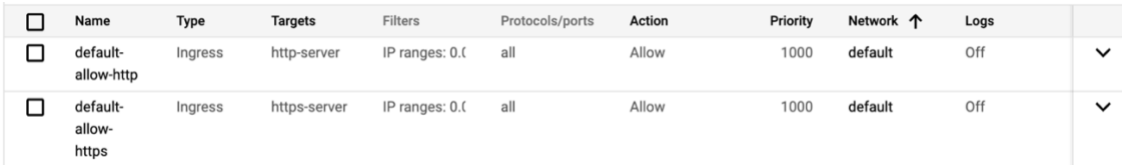
Followed the tutorials, Using GCP cloud account, configure and initialize Apache Spark cluster. I did the following steps:

1). Create a compute engine instance: an instance using E2 and Ubuntu 20.04LTS.

Name	Zone	Recommendation	In use by	Internal IP	External IP
data-assignment3	us-central1-a			10.128.0.2 (nic0)	34.122.48.230

Figure 1: Compute Engine Instance

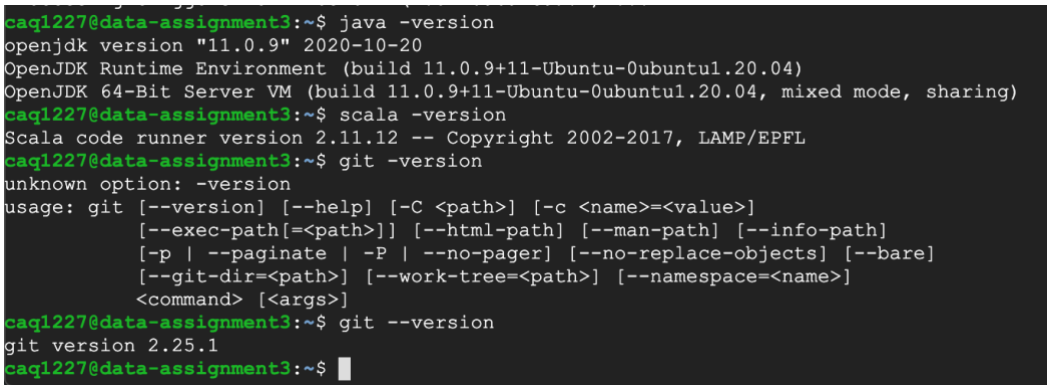
2). Change the protocols/ports to all.



Name	Type	Targets	Filters	Protocols/ports	Action	Priority	Network	Logs
default-allow-http	Ingress	http-server	IP ranges: 0.(	all	Allow	1000	default	Off
default-allow-https	Ingress	https-server	IP ranges: 0.(	all	Allow	1000	default	Off

Figure 2: Protocols/port is all

3). Download and set up the path for Python, pip, Scala, Apache Spark.



```
caq1227@data-assignment3:~$ java -version
openjdk version "11.0.9" 2020-10-20
OpenJDK Runtime Environment (build 11.0.9+11-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.9+11-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)
caq1227@data-assignment3:~$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
caq1227@data-assignment3:~$ git -version
unknown option: -version
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
caq1227@data-assignment3:~$ git --version
git version 2.25.1
caq1227@data-assignment3:~$
```

Figure 3: Check for versions

```
caq1227@data-assignment3:~$ mkdir Apache_Spark
caq1227@data-assignment3:~$ cd Apache_Spark
caq1227@data-assignment3:~/Apache_Spark$ wget https://httpd-mirror.sergal.org/apache/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz
--2020-11-10 14:43:46--  https://httpd-mirror.sergal.org/apache/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz
Resolving httpd-mirror.sergal.org (httpd-mirror.sergal.org)... 149.56.65.36, 2607:5300:60:494f::105
Connecting to httpd-mirror.sergal.org (httpd-mirror.sergal.org)|149.56.65.36|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 219929956 (210M) [application/x-gzip]
Saving to: 'spark-3.0.1-bin-hadoop2.7.tgz'

spark-3.0.1-bin-hadoop2.7.tg 100%[=========================================>] 209.74M  7.21MB/s    in 2m 35s

2020-11-10 14:46:21 (1.35 MB/s) - 'spark-3.0.1-bin-hadoop2.7.tgz' saved [219929956/219929956]
```

Figure 4: Download Spark and Export the Path

4). Using PySpark to initialize clusters

```
caq1227@data-assignment3:~$ pyspark
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.0.
1.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
20/11/13 21:38:30 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable
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).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Python version 3.8.5 (default, Jul 28 2020 12:59:40)
SparkSession available as 'spark'.
>>>
```

Figure 5: PySpark Running on GCP

2. Data Extraction and Preprocessing Engine

Tweets Extraction

Files on GitLab: https://git.cs.dal.ca/anqi/csci5408-a1-oceantracking/-/tree/master/A3

raw.py - the script to do searching and streaming and save tweets into Collections in RawDb

To run it on GCP: *eg. python raw.py -w Storm* (any keyword in the list)

1). tweepy.StreamListener is used to catch data with keyword on streaming.

streamer.filter(track=[KEYWORD], languages=["en"], stall_warnings=True) is called to track the live data with the given keyword and filter contents only in English.

Inside the StreamListener, using db[KEYWORD].insert_one(datajson) to save data into MongoDB one by one as JSON format. I also set the tweets limit as 200 (199) for streaming data about one word.

```
def on_data(self, data):

    try:
        client = MongoClient(MONGO_HOST)
        db = client.RawDb
        datajson = json.loads(data)
        self.num_tweets += 1
        if self.num_tweets < 200:
            id_str = datajson['id_str']
            db[KEYWORD].insert_one(datajson)
            print("Tweet with id: " + id_str + " saved in MongoDB.")
            return True
        else:
            return False
    except ProtocolError:
        print("")
    except Exception as e:
        print(e)
```

Figure 6: Core Code for Tweepy Streaming

2). To search related data, I used tweepy.Cursor

It is easy to search the data filtered by keyword query and language. Then the 200 data are also saved to the corresponding MongoDB Collection as the JSON format, each tweet in a document.

```
# Extracting date using the Search API
def save_search_mongodb():
    try:
        api = tweepy.API(auth)
        client = MongoClient(MONGO_HOST)
        db = client.RawDb
        collection = db[KEYWORD]
        for tweet in tweepy.Cursor(api.search, q=KEYWORD, lang='en').items(200):
            data = tweet._json
            id_str = data['id_str']
            collection.insert_one(data)
            print("Tweet with id: " + id_str + " saved in MongoDB.")
    except Exception as e:
        print(e)
```

Figure 7: Main Code for Tweepy Searching

Totally, 3192 tweets are extracted from searching and streaming half-and-half, 499 for every collection (word), displayed in Figure 8. The exported json files can be found in rawTweets folder, json files of data after cleaning are in cleanTweets folders.

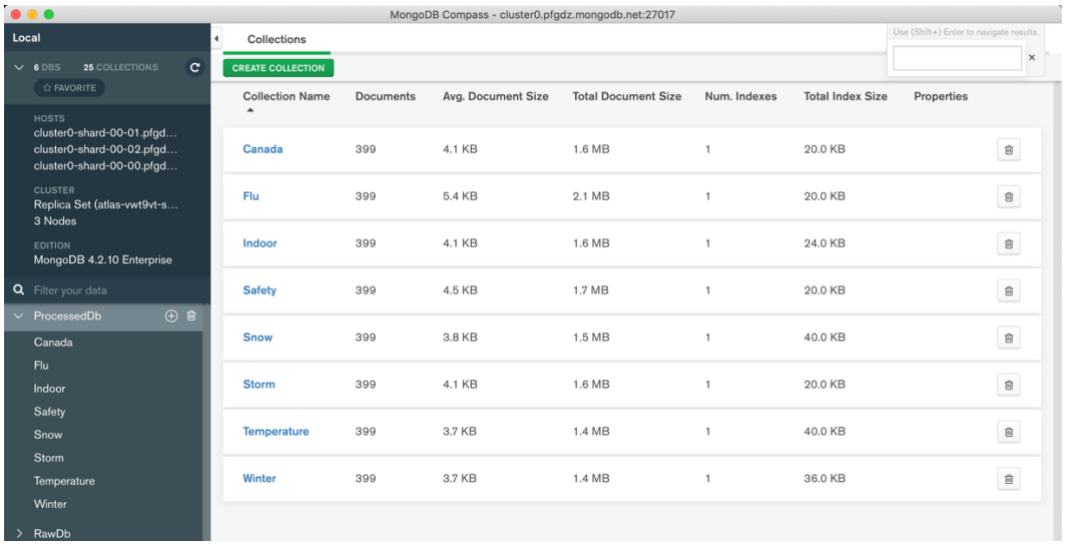


Figure 8: Data Extracted in RawDB

clean.py – the script to clean the raw data and save into Collections in ProcessedDb

The main purpose of this step is to make sure the data will not contain special characters, URLs, and emojis, prepared for future use.

To remove emoji, I first tried to filter out all the Unicode emojis used and match with Regex, but I found after using json_util.dumps(document) function, the new text is only contains the pure Unicode (like /uXXXX) for that emoji, not the original one.

The regex I use to remove URLs is “(https?|ftp|file)://[-A-Za-z0-9+&@#/%?=~_|!:,.;]+[-A-Za-z0-9+&@#/%=~_|]”. After cleaning, all data are saved into ProcessedDb (Figure 9).

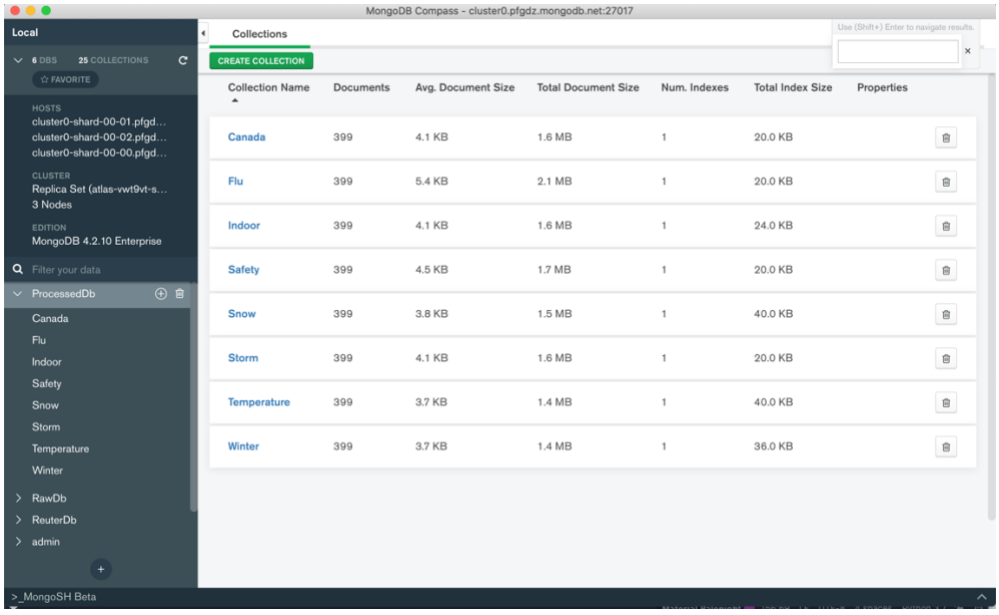


Figure 9: Data Saved in ProcessedDB

The cleaning results can be seen as the sample below (Figure 10 and 11):


```
_id: ObjectId("5fab2d63ae3e49047b64d0a1")
created_at: "Wed Nov 11 00:16:26 +0000 2020"
id: 1326317826420793344
id_str: "1326317826420793344"
text: "That's my President! 🇺🇸🇺🇸🇺🇸🇺🇸"
source: "<a href="http://twitter.com/download/android" rel="nofollow">Twitter f..."
truncated: false
```


Figure 10: Data before Cleaning


```
_id: ObjectId("5fab2d63ae3e49047b64d0a1")
created_at: "Wed Nov 11 00:16:26 +0000 2020"
id: 1326317826420793344
id_str: "1326317826420793344"
text: "That's my President! "
source: "<a href="" rel="nofollow">Twitter for Android</a>"
truncated: false
```


Figure 11: Data after Cleaning

Reuter News Articles Extraction

Reuter.py – the script to read and extract contents between three tags.

1). Use regex to match contents between two tags.

Eg. r"<REUTERS\b[^>]*>([\s\S]*?)(.*?)</REUTERS>"

2). Use re.findall() to loop through all occurrences.

3). Use ''.join() to convert the matching content into a string.

4). Within the loop, append three parts into the result list.

5). Save into MongoDB by using result list to construct the dictionary.

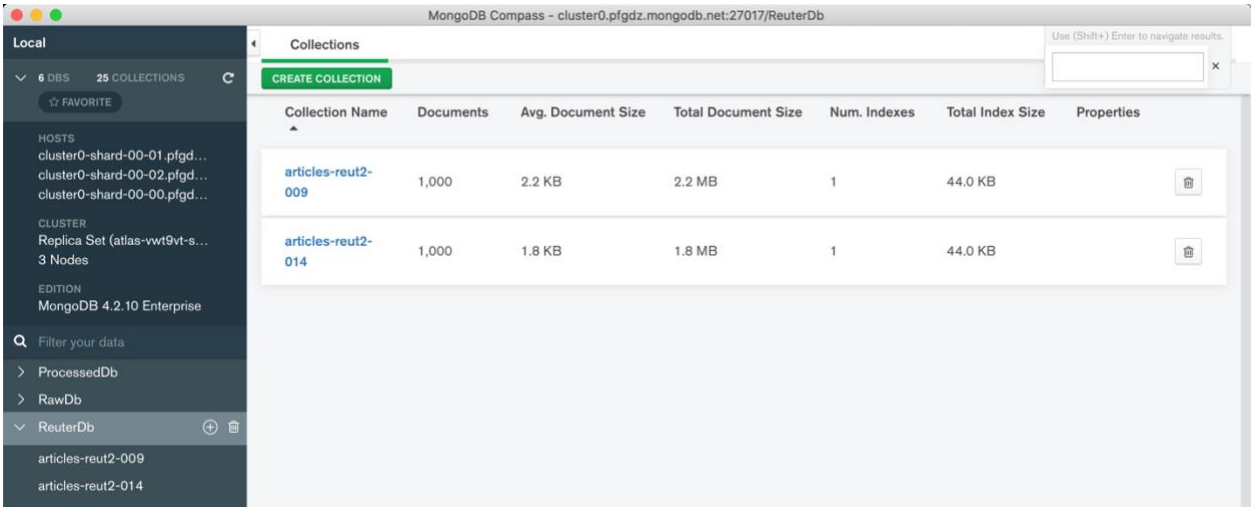


Figure 12: Data in ReutersDB

News articles are saved into ReuterDb, each collection (gathered from one .sgm file) contains 1000 documents which means 1000 articles in it (seen in Figure 12). The exported json files can be found in Reuters folder.

3. Data Processing using Spark

Process.py – the script to perform the frequency count on the stored ProcessedDb and ReuterDb.

To perform the word count, my MapReduce Program is implemented as follows:

1). Read several json files as input

spark.read.option("multiline", "true").json("cleanTweets/*.json")

I will now have all the data read into a spark data frame.

2). Clean data in the dataframe

To operate data in the dataframe, spark.sql is used to select related information. First filter the contents in a specific column into lower case, then remove punctuations in the text, later split each long text into rows of single word. Lastly having a loop through each row of the word dataframe, to see if it is the word in our list.

3) count the frequency of each word.

The result can be seen in the screenshots below. The word with highest frequency is “ice”; the one with lowest frequency is “cold”.

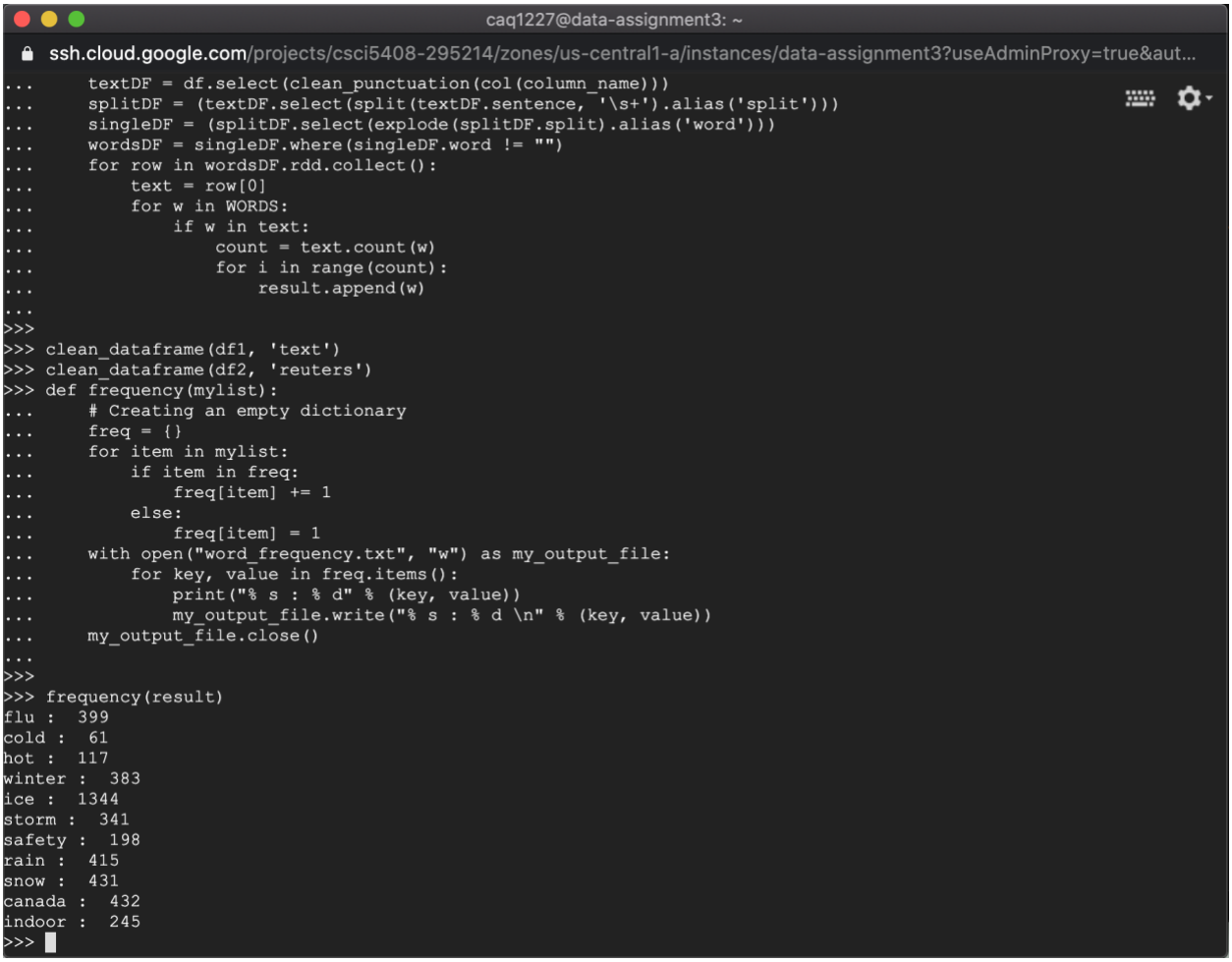


Figure 13: Process.py after running on GCP

4. Data Visualization using Graph Database

The cypher I used to create nodes and properties are like:

> CREATE (n: Season {name: "Winter", frequency: 383, related: "coldest season, snow and freezing temperature"})

The cypher I used to create relationships between two nodes are like:

> MATCH (a: Weather {}), (b: Condition {name: "Safety"}) MERGE (a) - [r: CARE] ->(b)

The generated graph is put below:

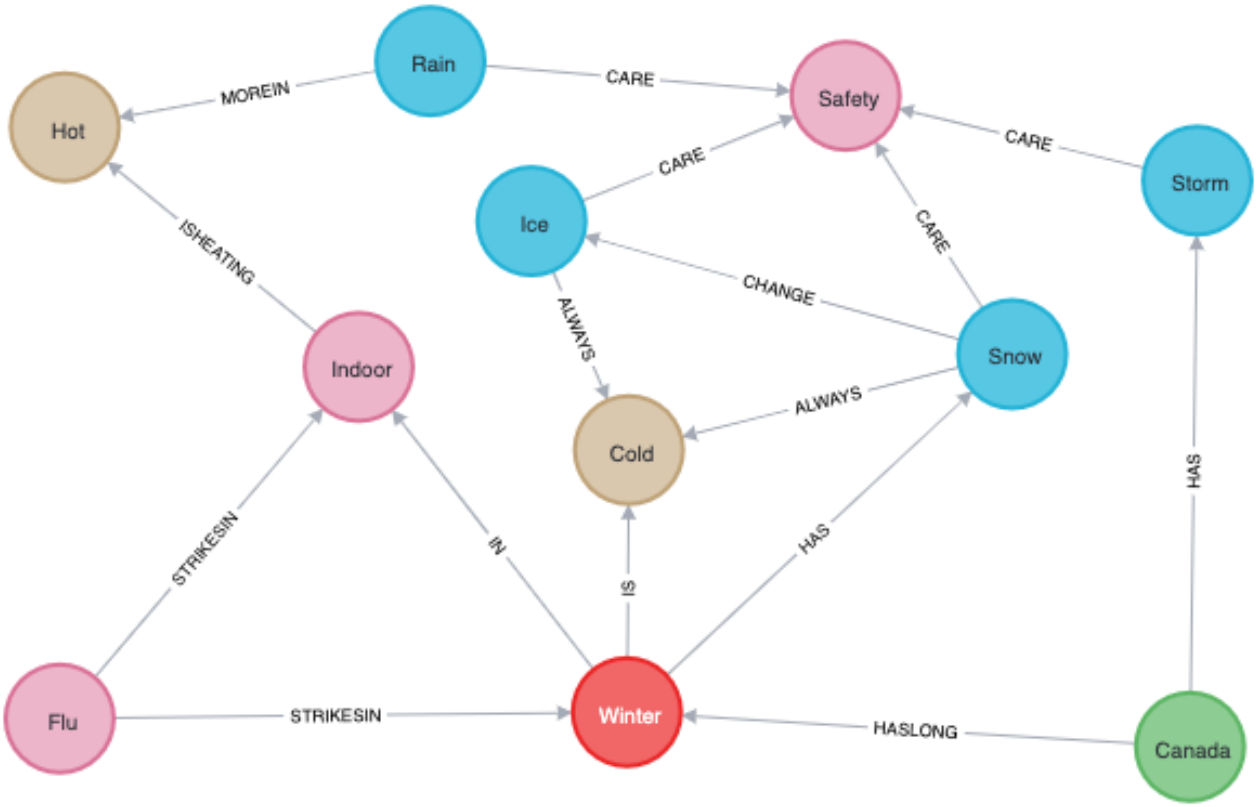


Figure 14: Nodes and Relationships Created by Neo4J

Reference

[1] “apache spark - how to loop through each row of dataFrame in pyspark,” *Stack Overflow*. https://stackoverflow.com/questions/36349281/how-to-loop-through-each-row-of-dataframe-in-pyspark [Accessed Nov. 13, 2020].

[2] K. Dhawan, “Big Data Analysis Using PySpark | Codementor.” https://www.codementor.io/@kunaldhawan93/big-data-analysis-using-pyspark-8y5yobb44 [Accessed Nov. 13, 2020].

[3] “MATCH - Neo4j Cypher Manual,” *Neo4j Graph Database Platform*. https://neo4j.com/cypher-manual/4.1/clauses/match/ [Accessed Nov. 13, 2020].

[4] Traversy Media, *Neo4j Graph Database & Cypher*. 2016. https://www.youtube.com/watch?v=1kyPUqU-MkE. [Accessed Nov. 13, 2020].

[5] “python - pyspark - multiple input files into one RDD and one output file,” *Stack Overflow*. https://stackoverflow.com/questions/35608326/pyspark-multiple-input-files-into-one-rdd-and-one-output-file [Accessed Nov. 13, 2020].

[6] “RegExr: Learn, Build, & Test RegEx,” *RegExr*. https://regexr.com/ [Accessed Nov. 13, 2020].

[7] “Spark SQL and DataFrames - Spark 3.0.1 Documentation.” http://spark.apache.org/docs/latest/sql-programming-guide.html [Accessed Nov. 13, 2020].

[8] “Streaming With Tweepy — tweepy 3.9.0 documentation.” http://docs.tweepy.org/en/latest/streaming_how_to.html [Accessed Nov. 13, 2020].

[9] “Tutorial — PyMongo 3.9.0 documentation.” https://api.mongodb.com/python/current/tutorial.html [Accessed Nov. 13, 2020].

[10] “word_count_dataframe - Databricks.” https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3328674740105987/4033840715400609/6441317451288404/latest.html [Accessed Nov. 13, 2020].