## HU Extension Assignment 07 E63 Big Data Analytics

### Handed out: 03/11/2017 Due by 9:30AM EST on Saturday, 03/25/2017

If you are familiar with NLP API-s in languages other than Python or Python NLP API-s other than NLTK please be free to solve these problems using technology of your choice.

**Problem 1.** Create atable displaying **relative** frequencies with which “modals” (can, could, may, might, will, would and should) are used in 18 texts provided by NLTK in the extract from Gutenberg Corpus.

Import Gutenberg corpus and list included books

|  |
| --- |
| from nltk.corpus import gutenberg  fileids = gutenberg.fileids()  for f in fileids:  print f |

|  |
| --- |
| austen-emma.txt  austen-persuasion.txt  austen-sense.txt  bible-kjv.txt  blake-poems.txt  bryant-stories.txt  burgess-busterbrown.txt  carroll-alice.txt  chesterton-ball.txt  chesterton-brown.txt  chesterton-thursday.txt  edgeworth-parents.txt  melville-moby\_dick.txt  milton-paradise.txt  shakespeare-caesar.txt  shakespeare-hamlet.txt  shakespeare-macbeth.txt  whitman-leaves.txt |

Calculate relative frequency for a list of modals verbs. Frequency is presented as % of total words on each text. Consider lower uppercase forms as same word

|  |
| --- |
| from nltk.probability import FreqDist  modals = ['can', 'could', 'may', 'might', 'will', 'would', 'should']  print " book\tcan\tcould\tmay\tmight\twill\twould\tshould"  print "---------------------------------------------------------------------------------------"  for f in gutenberg.fileids():  fdist = FreqDist([w.lower() for w in gutenberg.words(f)])  values=""  for m in modals:  values = values + "\t" + str(round(fdist.freq(m)\*100,2))  print "%25s" %f + values |

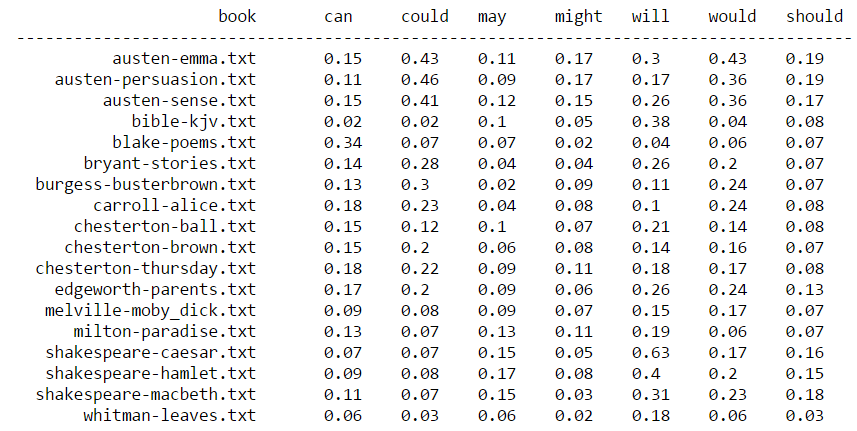


Figure 1 - Modal verb frequencies within Gutenberg's corpus

Find the most used modal and the least used modal in every examined text.

Added a new column that points out the most used modal on each text. Another for the least used

|  |
| --- |
| from nltk.probability import FreqDist  modals = ['can', 'could', 'may', 'might', 'will', 'would', 'should']  print " book\tcan\tcould\tmay\tmight\twill\twould\tshould\t most freq\t least freq"  print "--------------------------------------------------------------------------------------------------------------------"  for f in gutenberg.fileids():  fdist = FreqDist([w.lower() for w in gutenberg.words(f)])  values=""  mostFreq = 0.00  leastFreq = 1.00  for m in modals:  freqDist= fdist.freq(m)  if mostFreq < freqDist:  mostFreq = freqDist  mostModal = m  if leastFreq > freqDist:  leastFreq = freqDist  leastModal = m    values = values + "\t" + str(round(fdist.freq(m)\*100,2))  print "%25s" %f + values + "\t" + "%10s" %mostModal + "%10s" %leastModal |

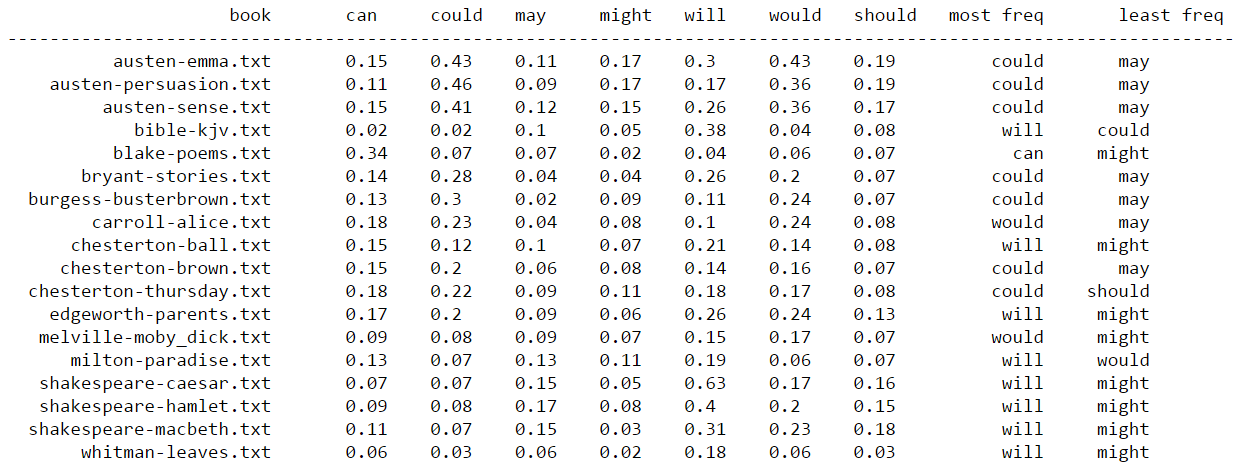


Figure 2 – Most frequent modal verb within Gutenberg's corpus

Select two text which use previously identified most frequently used modals the most. Compare usage in both texts by examining the concordances of that modals. Perhaps try to understand how the word is used in different texts.

The 2 first texts have the largest frequency for could out of all modals in all texts. I will be using those 2 texts austen-emma.txt and austen-persuasion.txt with could

|  |
| --- |
| from nltk import Text  emma = Text(gutenberg.words('austen-emma.txt'))  persuasion = Text(gutenberg.words('austen-persuasion.txt'))  emma.concordance('could')  persuasion.concordance('could') |

**Problem 2**. In the Inaugural corpus identify 10 most frequently used words longer than 7 characters.

|  |
| --- |
| from nltk.corpus import inaugural  fileids = inaugural.fileids()  for f in fileids:  print f |

|  |  |  |
| --- | --- | --- |
| 1789-Washington.txt  1793-Washington.txt  1797-Adams.txt  1801-Jefferson.txt  1805-Jefferson.txt  1809-Madison.txt  1813-Madison.txt  1817-Monroe.txt  1821-Monroe.txt  1825-Adams.txt  1829-Jackson.txt  1833-Jackson.txt  1837-VanBuren.txt  1841-Harrison.txt  1845-Polk.txt  1849-Taylor.txt  1853-Pierce.txt  1857-Buchanan.txt  1861-Lincoln.txt | 1865-Lincoln.txt  1869-Grant.txt  1873-Grant.txt  1877-Hayes.txt  1881-Garfield.txt  1885-Cleveland.txt  1889-Harrison.txt  1893-Cleveland.txt  1897-McKinley.txt  1901-McKinley.txt  1905-Roosevelt.txt  1909-Taft.txt  1913-Wilson.txt  1917-Wilson.txt  1921-Harding.txt  1925-Coolidge.txt  1929-Hoover.txt  1933-Roosevelt.txt  1937-Roosevelt.txt | 1941-Roosevelt.txt  1945-Roosevelt.txt  1949-Truman.txt  1953-Eisenhower.txt  1957-Eisenhower.txt  1961-Kennedy.txt  1965-Johnson.txt  1969-Nixon.txt  1973-Nixon.txt  1977-Carter.txt  1981-Reagan.txt  1985-Reagan.txt  1989-Bush.txt  1993-Clinton.txt  1997-Clinton.txt  2001-Bush.txt  2005-Bush.txt  2009-Obama.txt |

|  |
| --- |
| for f in inaugural.fileids():  words = words + [w.lower() for w in inaugural.words(f) if len(w)>7]  fdist = FreqDist([w for w in words])  fdist.plot(10) |

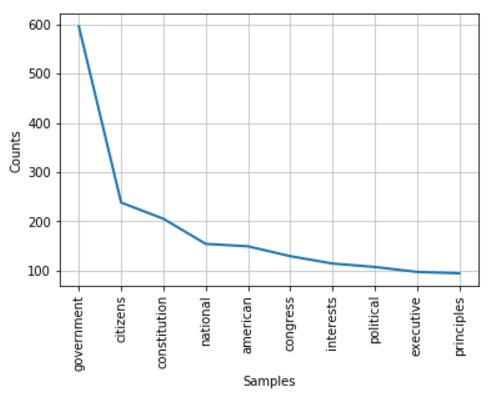


Figure 3 - Count of the 10 most common words in all the presidential inaugural corpus

Which one of those has the largest number of synonyms?

|  |
| --- |
| from nltk.corpus import wordnet as wn  syn\_list = []  for m in most\_common:  syn = wn.synsets(m)  count = 0  for s in syn:  s.lemma\_names()  count = count + len(s.lemma\_names())  print m + ":" + str(count) |

|  |
| --- |
| government:12  citizens:1  constitution:20  national:12  american:7  congress:17  interests:21  political:3  executive:6  principles:10 |

According to this, constitution is the word with the largets number of synomyms.

List all synonyms for those 10 words.

|  |
| --- |
| for m in most\_common:  syn = wn.synsets(m)  count = 0  print "::::::::::::::::::::" + m + "::::::::::::::::::::"  for s in syn:  print s.lemma\_names() |

|  |
| --- |
| ::::::::::::::::::::government::::::::::::::::::::  [u'government', u'authorities', u'regime']  [u'government', u'governing', u'governance', u'government\_activity', u'administration']  [u'government']  [u'politics', u'political\_science', u'government']  ::::::::::::::::::::citizens::::::::::::::::::::  [u'citizen']  ::::::::::::::::::::constitution::::::::::::::::::::  [u'fundamental\_law', u'organic\_law', u'constitution']  [u'constitution', u'establishment', u'formation', u'organization', u'organisation']  [u'United\_States\_Constitution', u'U.S.\_Constitution', u'US\_Constitution', u'Constitution', u'Constitution\_of\_the\_United\_States']  [u'constitution', u'composition', u'physical\_composition', u'makeup', u'make-up']  [u'Constitution', u'Old\_Ironsides']  ::::::::::::::::::::national::::::::::::::::::::  [u'national', u'subject']  [u'national']  [u'national']  [u'national']  [u'national']  [u'home', u'interior', u'internal', u'national']  [u'national']  [u'national']  ::::::::::::::::::::american::::::::::::::::::::  [u'American']  [u'American\_English', u'American\_language', u'American']  [u'American']  [u'American']  [u'American']  ::::::::::::::::::::congress::::::::::::::::::::  [u'Congress', u'United\_States\_Congress', u'U.S.\_Congress', u'US\_Congress']  [u'congress']  [u'congress']  [u'sexual\_intercourse', u'intercourse', u'sex\_act', u'copulation', u'coitus', u'coition', u'sexual\_congress', u'congress', u'sexual\_relation', u'relation', u'carnal\_knowledge']  ::::::::::::::::::::interests::::::::::::::::::::  [u'interest', u'involvement']  [u'sake', u'interest']  [u'interest', u'interestingness']  [u'interest']  [u'interest', u'stake']  [u'interest', u'interest\_group']  [u'pastime', u'interest', u'pursuit']  [u'interest']  [u'concern', u'interest', u'occupy', u'worry']  [u'matter\_to', u'interest']  ::::::::::::::::::::political::::::::::::::::::::  [u'political']  [u'political']  [u'political']  ::::::::::::::::::::executive::::::::::::::::::::  [u'executive', u'executive\_director']  [u'executive']  [u'administrator', u'executive']  [u'executive']  ::::::::::::::::::::principles::::::::::::::::::::  [u'principle', u'rule']  [u'principle']  [u'principle']  [u'principle', u'rule']  [u'principle', u'precept']  [u'rationale', u'principle'] |

Which one of those 10 words has the largest number of hyponyms?

|  |
| --- |
| hyp\_list = []  for m in most\_common:  syn = wn.synsets(m)  count = 0  for s in syn:  hyp = s.hyponyms()  for h in hyp:  count = count + len(h.lemma\_names())  print m + ":" + str(count) |

List all hyponyms of those 10 most frequently used “long” words.

|  |
| --- |
| government:32  citizens:9  constitution:17  national:4  american:114  congress:18  interests:45  political:0  executive:19  principles:64 |

According to the above, American has the largest number of hyponyms.

|  |
| --- |
| for m in most\_common:  syn = wn.synsets(m)  count = 0  print "::::::::::::::::::::" + m + "::::::::::::::::::::"  for s in syn:  hyp = s.hyponyms()  for h in hyp:  print h.lemma\_names() |

The purpose of this problem is to familiarize you with WordNet and concepts of synonyms and hyponims.

Your literature for Problems 1 and 2 are chapters 1 and 2 of Natural Language Processing with Python book by Steven Bird et al.

**Problem 3.** Create your own grammar for the following sentence:

“Describe every step of your work and present all intermediate and final results in a Word document”. Present the syntactic structure of that sentence as a tree. You could use Ne04J to create the visual graph of that tree. You graph does not have to look exactly like the graphs in the book. You are welcome to use any other technique or API to create that graph

All the process has been done within Jupyter notebooks

|  |
| --- |
| from nltk import RecursiveDescentParser  from nltk.tree import Tree  from nltk import CFG  from nltk import word\_tokenize  #grammar definition  grammar = CFG.fromstring("""  S -> S1 C S1  S1 -> VP NP | VP NP PP  VP -> V | C V  NP -> Det N P Det N | Det AP N  PP -> P Det Adj N  AP -> Adj C Adj  N ->'step' | 'work' | 'results' | 'document'  Adj -> 'intermediate' | 'final' | 'Word'  Det -> 'every' | 'your' | 'all' | 'a'  V -> 'Describe' | 'present'  P -> 'of' | 'in'  C -> 'and'  """)    sent = word\_tokenize("Describe every step of your work and present all intermediate and final results in a Word document")    #parse sentece according to grammar  parser = RecursiveDescentParser(grammar)  trees = parser.parse(sent)  i=0  for tree in trees:  print tree  t = Tree.fromstring(str(tree))  #tree representation  t.pretty\_print()  i = i + 1  if i == 1:  break |

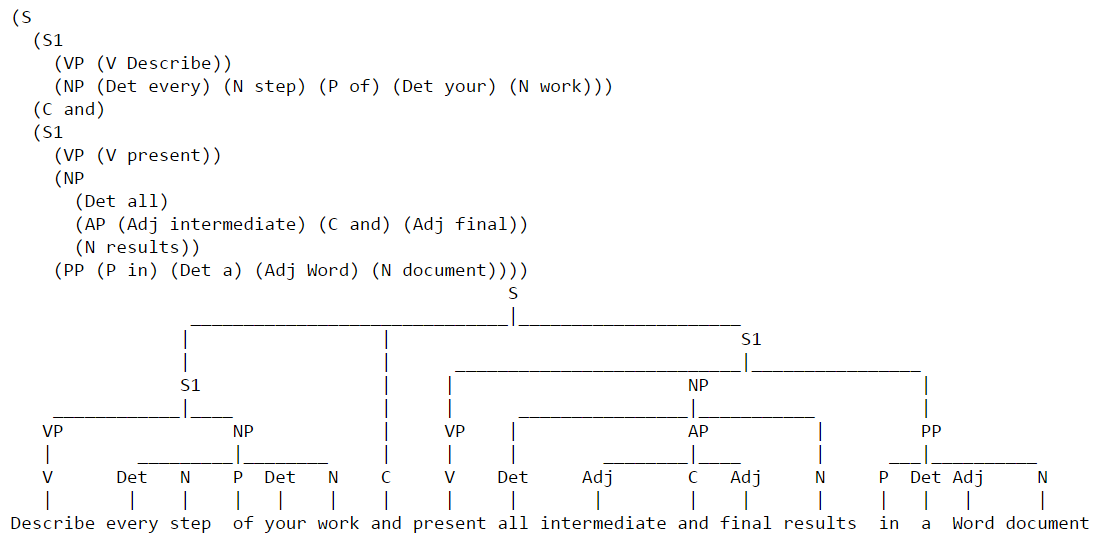


Figure 4 - Sentence representation as per new grammar definition

Your literature for Problem 3 is chapter 8 of Natural Language Processing with Python book by Steven Bird et al.

**Problem 4.** Install Cassandra server on your Cloudera VM or any other VM if you so prefer. Use one of the methods described in the lab.

I’m using a CentOS 7 machine. No Java is installed so first thing I install Java:

|  |
| --- |
| sudo yum install java-1.7.0-openjdk |

Download Cassandra tar file, decompress it and copy its content to a new directory named cassandra:

|  |
| --- |
| wget <http://supergsego.com/apache/cassandra/2.1.17/apache-cassandra-2.1.17-bin.tar.gz>  tar zxvf apache-cassandra-2.1.17-bin.tar.gz  mkdir cassandra  mv apache-cassandra-2.1.17/\* cassandra/ |

Edit cassandra.yaml file, and uncomment this 3 lines:

|  |
| --- |
| data\_file\_directories:/var/lib/cassandra/data  commitlog\_directory: /var/lib/cassandra/commitlog  saved\_caches\_directory: /var/lib/cassandra/saved\_caches |

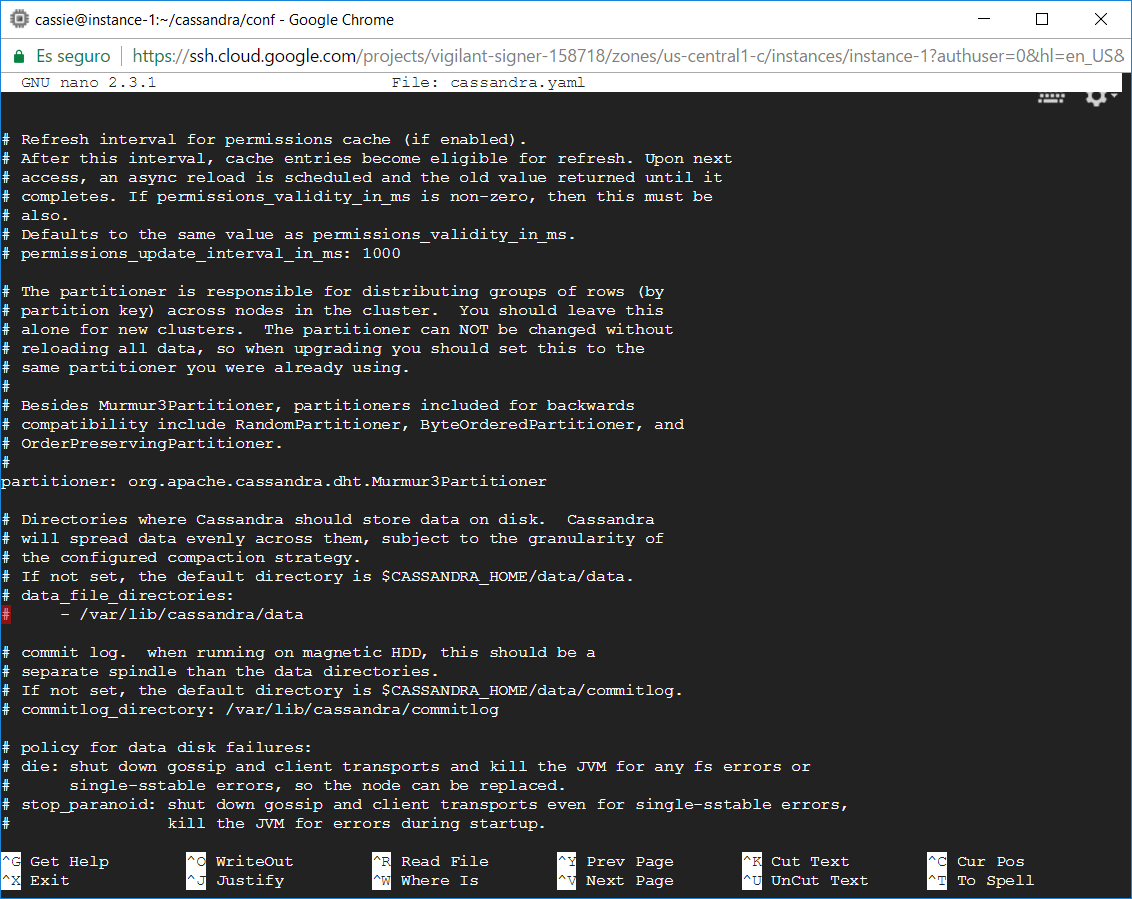


Figure 5 - cassandra.yaml config file

According to the paths defined in config.yaml fil create the required folders and grant permissions to all users:

|  |
| --- |
| sudo mkdir /var/lib/cassandra  sudo mkdir /var/log/cassandra  sudo chmod 777 /var/lib/cassandra/  sudo chmod 777 /var/log/cassandra/ |

Now start Cassandra:

|  |
| --- |
| ./cassandra -f |

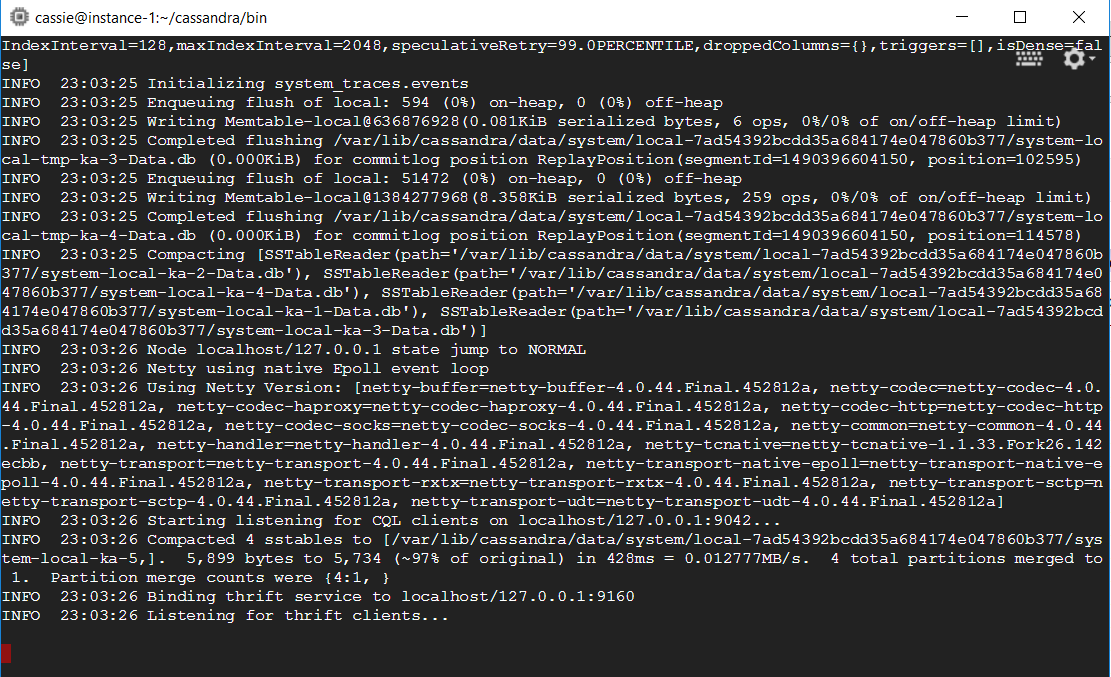


Figure 6 - Cassandra view from console

Use Cassandra SQL Client, cqlsh, to create and populate table person.

First connect to Cassandra

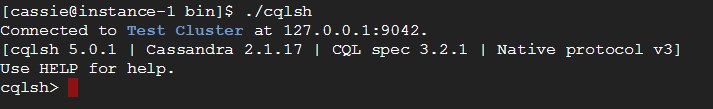


Figure 7 - Connection to cassandra from Linux console

Let every person by described by his or her first and last name, and city where he or she lives. Let every person possess up to three cell phones.

Afterwards create table

|  |
| --- |
| CREATE TABLE person (  person\_id int,  first\_name text,  last\_name text,  city text,  cell\_1 text,  cell\_2 text,  cell\_3 text,  PRIMARY KEY (person\_id)); |

Populate your table with three individuals using cqlsh client. Demonstrate that you can select the content of your table person.

Next image shows insert sentences and select sentence with its result after the 3 record creation

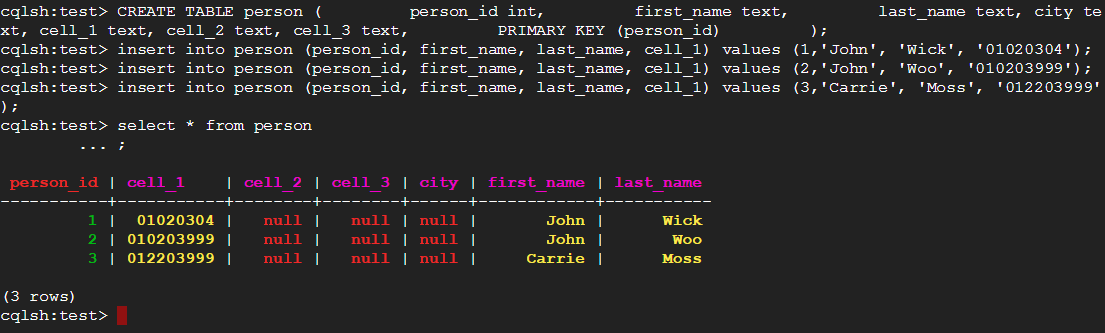


Figure 8 - Create records in Cassandra's person table

Write a simple client in a language of your choice that will populate 2 rows in Casandra’s table person, subsequently update one of those rows, for example change the city where a person lives, and finally retrieve that modify row from Cassandra and write its content to the console.

I will work with Python, therefor is to install Cassandra’s driver for python to be able to connect from Python to our database

|  |
| --- |
| sudo yum install python-pip  sudo pip install cassandra-driver  python |

The following code produces what requested. It creates 2 new records on person’s table, it updates city value for one of them (from London to Manchester) and lists the result for that user within the console

|  |
| --- |
| from cassandra.cluster import Cluster  cluster = Cluster()  session = cluster.connect('test')  session.execute("INSERT INTO person (person\_id, first\_name, last\_name, city, cell\_1, cell\_2, cell\_3) VALUES (4, 'tony', 'blair', 'london', '111', '2222', '3333')")  session.execute("INSERT INTO person (person\_id, first\_name, last\_name, city, cell\_1, cell\_2, cell\_3) VALUES (5, 'margaret', 'thatcher', 'london', '444', '5555', '6666')")  session.execute("UPDATE person set city = 'manchester' where person\_id=5")  result = session.execute("SELECT \* FROM person WHERE person\_id=5")  row = result[0]  print row |

The below screenshot shows the result:

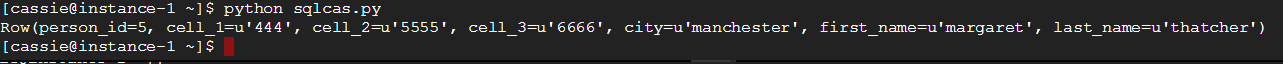


Figure 9 - Script execution console output

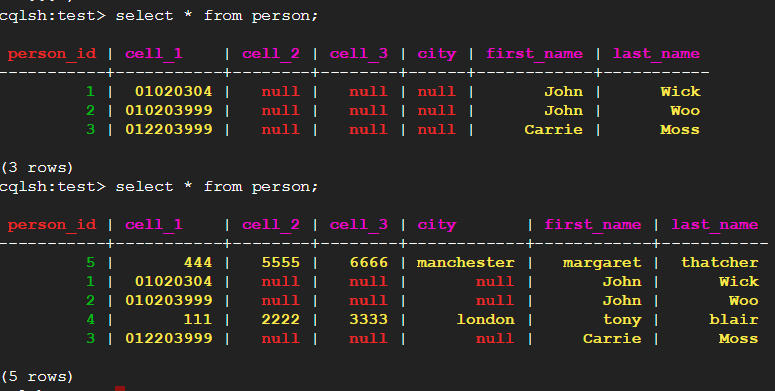


Figure 10 - Table person after running required python code

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. If you use some other language other than Python in your daily work with NLP, please be free to use that language and a framework of your choice to do this assignment.