PROJECT ASSIGNMENT I

UNIVERSITY KNOWLEDGE GRAPH

A Report
Presented to
The Department of Electrical & Computer Engineering
Concordia University

In Partial Fulfillment
of the Requirements
of COMP 474/6741

by

Filip Jodoin	ID: 27879105
Ling Zhi Mo	ID: 40024810
Sean O'Connor	ID: 40016165

Concordia University
March 9th 2020

ABSTRACT:

The first phase of our project is to gather information on all Concordia University courses. This information will be used to generate a Knowledge Graph. This report will discuss the details regarding the three steps (obtaining, shaping and testing) data to reach the deliverables assigned for this phase.

OBJECTIVES:

Creating a Knowledge Graph with a dataset complete with all available courses (along with their respective details) at Concordia University. A Querying section must be implemented in order to test the validity of our Knowledge Graph.

TASKS:

Vocabulary

Description how you modeled the schema for your knowledge base, including the vocabularies you reused, any vocabulary extensions you developed, etc. Give brief justifications where appropriate (e.g., choice of existing vocabulary).

We made use of existing vocabularies as listed below.

Table 1: Vocabulary

RDF	type	For defining classes and properties
	Property	
RDFS	Class	For defining classes
	subClassOf	For extending class definitions
	domain	For defining properties
	range	
FOAF	firstName	For the personal information of students in our knowledge base
	lastName	
	mbox	
	Person	A student is a person
	topic	Property used to relate a topic to a university course
DBO	document	For defining our transcripts document seemed a fitting concept.
	type	For use of the Public_university type
DBR	Course_(education)	Suitable for broadly defining the courses offered at universities
	Public_university	Suitable definition for Universities
XSD	string	Literal strings serve as the object of several properties we have generated.

We re-used the focu schema <http://focu.io/schema#> from the lab exercises contain our classes and properties and the ex schema <http://example.org/> to contain instances of these classes. Their definitions and details concerning their usage are given below. As per the assignment specifications each class and property created for our vocabulary possesses an rdfs:label and rdfs:comment which are omitted for brevity.

focu:Student (class):

rdf:type *rdfs:Class*
rdfs:subClassOf *foaf:Person*

An entity defined by us to represent students in our knowledge base. Based off of similar ones created during the lab exercises and in-class worksheets.

ex:Student (instance):

rdf:type *focu:Student*	foaf:firstName
focu:hasRecord	foaf:lastName
	foaf:mbox

Students possess URIs of the format <http://example.org/X>, where X is an integer value generated sequentially during knowledge base creation. Students possess first names, last names and email addresses as defined in the FOAF vocabulary. We define an additional property *hasRecord* (described below). **A student may have 0 or more hasRecord predicates**, one per *Record*.

focu:hasRecord (property):

rdf:type *rdf:Property*	rdfs:domain focu:Student
	rdfs:range focu:Record

A property defined by us to associate a single record to a single student. Refer to *Record* below for more information.

focu:subject **focu:catalog** **focu:subject_catalog**

The Concordia opendata API from which we retrieve course information returns the discipline of the course (i.e. COMP) and the number of the course (i.e. 474) as subject and catalog, respectively. Each of these three terms are properties possessed by other classes in our knowledge base. All three are kept separate as they are needed or convenient when querying or navigating our graph.

focu:Record (class):

rdf:type *dbo:document*

An entity representing a single transcript for a particular class. Refer to the **instance** definition below for more information.

ex:Record (instance):

rdf:type *focu:Record*	focu:semester
	focu:subject_catalog
	focu:grade

Records possess URIs of the format <http://example.org/recordX>, where X is an integer value generated sequentially during knowledge base creation. Record instances possess actual data for a given record associated grades (A, B, … , F) the subject_catalog (i.e. COMP474) and a semester (annual i.e. 2001, 2002, … , 2019). Each of these data points are defined in focu as properties possessed by Records.

ex:Course (instance):

rdf:type *dbo:Course_(education)*	focu:subject
foaf:topic	focu:catalog

Our course instances are identified by the combination of their subject (i.e. COMP) and catalog (i.e. 474) to create their URI <http://example.org/COMP474>. The terms subject and catalog were detailed previously in this report. **A course may have 0 or more foaf:topic predicates**, one for each topic. Topics are discovered by dbpedia Spotlight using the course description and title to offer more information about a course.

Knowledge Base Construction

Describe (a) your dataset and (b) your process and developed tools for populating the knowledge base from the dataset. Describe how to run the tools to create the knowledge base. Explain your process for linking entities to DBpedia.

(a) Describe dataset

- The dataset is stored in a local file “database.txt”.
- The dataset contains 7050 data points, where each datapoint contains information on a course from Concordia University.
- The datapoints are in a string dictionary format, as per our request (during API call process; consult (b), Database_builder.py section below)

- Each datapoint has the following dictionary structure:

  i.e.

```
{
   "000026":{
      "title":"Financial and Managerial Accounting",
      "subject":"ACCO",
      "catalog":"220",
      "career":"UGRD",
      "classUnit":"3.00",
      "prerequisites":"Prerequisites of this course",
      "crosslisted":"null"
      "description":"Description of this course",
      "sub_catalog":"ACCO 220"
   }
}
```

(b) Describe developed tools

- database_builder.py (Building dataset)
  - **<u>Step 1 (line 23):</u>** CONCORDIA'S OPEN DATA api call (api_call())
    - In this step, we are using Concordia's OpenData api to gather all available course information. Specifically, we are requesting "Course Description" which contains a course unique id with the course description, and "Course Catalog" which contains the same course unique id with its catalog information (i.e course title, course subject, course number, etc.). To accomplish this http request, we used python module "requests". Once the api call returns the results string, we will build the dictionary objects (the datapoint), and dump the dictionary to the local file "database.txt".
  - **<u>Step 2 (line 125):</u>** Crawling Concordia's undergraduate calendar (undergrad_scheme())
    - In this step, we are updating our dataset for the datapoints that are undergraduate courses and are missing a specific "description". To accomplish this goal, we first used "requests" to get Concordia's undergraduate program's main page, which contains all subjects' undergraduate calendar webpage that has course description. Next, we will request each url by using python module "requests_futures", an extension module of "request", and parse the return web content to retrieve the missing undergraduate course description. Finally, we will update our database file with the new dataset.

- ➢ **Step 3 (line 194):** Crawling Concordia's graduate calendar (graduate_scheme())
  - In this step, we are updating our dataset for the datapoints that are graduate courses and are missing a specific "description". To accomplish this goal, we first used "requests" to get Concordia's graduate program summary page, which contains every graduate program main page and corresponding graduate program course calendar page that has a graduate course description. Similar to the above step, we used "requests_futures" to make a http request to each program calendar, retrieve required information and update our database.

❖ parse.py

- ➢ **Step 1 (line 61):** Extract relevant entries from each database.txt datapoint
  - Our project's scope only requires certain fields for each course, therefore it is important to not include unnecessary information into our knowledge graph.
- ➢ **Step 2 (line 89):** Obtain Topics related to each course
  - In order to obtain all topics related to a course, the title and description of said course must be passed to DBpedia Spotlight. DBpedia Spotlight is hosted locally through docker and will return all tagged topics which were generated. If a course DOES NOT return a topic, it is deemed irrelevant and is discarded.
- ➢ **Step 3 (line 135):** Add all valid courses to Knowledge Graph
  - Now that all the valid courses have been parsed and separated from the invalid and unstructured courses, we must populate the Knowledge Graph.
- ➢ **Step 4 (line 160):** Generating Students and adding them to Knowledge Graph
  - In this step, we used a custom function (create_students() line 23) that will generate 100 students where each one has a unique id, first name, last name, email address and a randomized record list. Each record in the record list contains the following information: year of complete, course (subject and catalog), and corresponding grade.

Queries

- query.py

Describe your translation of the knowledge base queries below into SPARQL. Provide (brief) example output for each query.

The following image is the menu in the query.py module;

```
Enter 1 for number of triple in Knowledge Base
Enter 2 for total number of students, courses and topics
Enter 3 for all topics related to a given course
Enter 4 for all completed courses for a given student
Enter 5 for all students familiar with a given topic
Enter 6 for all topics familiar to a given student
 >
```

Figure 1: query.py menu

1. Total number of triples in the KB

```
 > 1
Number of triple(s) in Knowledge Base:  49619
```

Figure 2: Query 1 output; Number of Triples in Knowledge Graph

- Line 33; simply prints the length of our graph g. No SPARQL query required.

2. Total number of students, courses, and topics

```
 > 2
100 Students
4595 Courses
6482  Topics
```

Figure 3: Query 2 output; number of Students, Courses and Topics in Knowledge Graph

- Line 38; Performs 3 different SPARQL queries. Essentially, the Students and Courses use the COUNT(*) AS ?[variable_name] keyword in the SPARQL query to obtain their values. The Topics count query first populates a list with all topics obtained from its SPARQL query; leverages DISTINCT and GROUP BY to ensure the topics list is populated with unique entries. The length of this list is the number of Topics in our graph.

3. For a course c, list all covered topics using their (English) labels and their link to DBpedia

```
 > 3
Please enter the Subject Catalog (i.e. ACCO435) of a Course
 > COMP474

- Topic Label:
College of Osteopathic Medicine of the Pacific
- Dbpedia URL:
 http://dbpedia.org/resource/College_of_Osteopathic_Medicine_of_the_Pacific

- Topic Label:
Artificial intelligence
- Dbpedia URL:
 http://dbpedia.org/resource/Artificial_intelligence

- Topic Label:
Computer simulation
- Dbpedia URL:
 http://dbpedia.org/resource/Computer_simulation

- Topic Label:
Expert system
- Dbpedia URL:
 http://dbpedia.org/resource/Expert_system

- Topic Label:
Blackboard
- Dbpedia URL:
 http://dbpedia.org/resource/Blackboard

- Topic Label:
Coen River
- Dbpedia URL:
 http://dbpedia.org/resource/Coen_River

- Topic Label:
Conflict resolution
- Dbpedia URL:
 http://dbpedia.org/resource/Conflict_resolution

- Topic Label:
Knowledge acquisition
- Dbpedia URL:
 http://dbpedia.org/resource/Knowledge_acquisition
```

Figure 4: Query 3 output; Topics and DBpedia URLs associate with a given course (subject + catalog)

- Line 77; With the course (subject + catalog) given as an input, the SPARQL query is done very fast. For the Topics, it only needs to search for all rdf:topic associated with the course. The Labels are obtained by parsing the rdf:topic obtained previously. Essentially, we would attach a Label to each Topic in the Knowledge Graph by calling the DBpedia spotlight candidate API option… unfortunately it is not included in the docker image we have hosting, therefore we have not found an efficient way to fetch the labels via API without getting blocked.

4. For a given student, list all courses this student completed, together with the grade

```
 > 4
Please enter the ID of a given Student
 > 78
Complete Courses for [Student 78]
2010 http://example.org/MAST898C D
2016 http://example.org/ENGL252 B
2009 http://example.org/DART262 A
2003 http://example.org/CEPS1024E B
2017 http://example.org/ELEC6391 B
1998 http://example.org/HISW242 B
1991 http://example.org/EXCI357 A
2011 http://example.org/DRAW410 C
2008 http://example.org/MANA298 A
1995 http://example.org/FBRS260 C
```

Figure 5: Query 4 output; courses (semester, subject+catalog and grade) completed by given student id

- Line 100; By giving the query a student id, the SPARQL query will first query for the records of the given student. Each record contains a semester, a course (subject + catalog) and a grade; said information is retrieved and only printed if the grade obtained is NOT “F”.

5. For a given topic, list all students that are familiar with the topic (i.e., took, and did not fail, a course that covered the topic)

```
 > 5
Please enter the topic of interest
 > Psychology
http://example.org/64
http://example.org/35
http://example.org/100
http://example.org/67
http://example.org/98
http://example.org/22
http://example.org/23
http://example.org/35
http://example.org/42
http://example.org/22
http://example.org/90
http://example.org/6
http://example.org/35
http://example.org/25
http://example.org/32
http://example.org/11
```

Figure 6: Query 5 output; return student id of students who are familiar with a given Topic

- Line 124; The SPARQL query starts by obtaining the course (subject + catalog) associated with the given Topic (input). All of the records connected to said course

are then fetched. Records with grades which are NOT “F” (by using FILTER) will be kept for a final query; return student id of belonging to records kept.

6. For a student, list all topics (no duplicates) that this student is familiar with (based on the completed courses for this student that are better than an “F” grade)

```
 > 6
Please enter the ID of a given student
 > 45
http://dbpedia.org/resource/Hebrew_language
http://dbpedia.org/resource/Classical_music
http://dbpedia.org/resource/Syntax
http://dbpedia.org/resource/Academic_grading_in_the_United_States
http://dbpedia.org/resource/Economic_efficiency
http://dbpedia.org/resource/Monopoly
http://dbpedia.org/resource/Microeconomics
http://dbpedia.org/resource/Satire
http://dbpedia.org/resource/Maria_Edgeworth
http://dbpedia.org/resource/Romance_novel
http://dbpedia.org/resource/Ann_Radcliffe
http://dbpedia.org/resource/Charles_Burney
http://dbpedia.org/resource/Charles_Dickens
http://dbpedia.org/resource/Literary_genre
http://dbpedia.org/resource/Gothic_fiction
http://dbpedia.org/resource/Literary_realism
http://dbpedia.org/resource/Christian
http://dbpedia.org/resource/Jesus
http://dbpedia.org/resource/Bible
```

Figure 7: Query 6 output; Display all Topics familiar to given student id

- Line 148; SPARQL query obtains all records associated with the student id given. If a record’s grade is not “F”, then it proceeds to the next query. From here, SPARQL will extract the course (subject + catalog) of said record. Finally, all topics linked to the course will be considered as familiar topics.

APPENDIX:

Assumptions:

- Concerning the 5th element of the Courses deliverable;

  5. A link (rdfs:seeAlso) to a web page with the course information, if available

  Our data was sourced using the Concordia *opendata.concordia.ca/API* which does not provide an additional link for courses, therefore this task is not covered in our report.