Homework #2: SPARQL

The following exercises are due at the beginning of class on Wednesday, Feb. 19. There are two sections: written exercises and an electronic exercise. This will count for 10% of your overall grade.

Written Exercises:
The exercises in this section should be completed and turned in on paper.

1. *[20 pts.]* For this exercise, assume that you are querying DBPedia (dbpedia.org). The ontology is defined by http://dbpedia.org/ontology/; I recommend viewing the Turtle format, as the default HTML only provides a partial list of the classes and properties. Write the following queries in SPARQL syntax. You may find it helpful to test your queries using the SPARQL endpoint (http://dbpedia.org/sparql/), and exploring some of the instances to see how the data is actually structured.
   a) Retrieve the names of all universities and the URI of the country that each is located in.
   b) For each country founded between 1900 and 1980 (inclusive), retrieve its name, the date it was founded, and if it was dissolved, retrieve the year of dissolution (see dbo:dissolutionYear) as well. Sort the answers by founding date. Note, DBPedia has many things that are not actually countries recorded as countries; you do not need to eliminate such answers. Hint: For SPARQL to perform the comparisons correctly, make sure that you explicitly specify that your constants are dates.
   c) Find all companies with "soft" in the name whose industry matches the DPBedia resources for the "Internet" or "Software" industries. Return the name and (if known) the number of employees (dbo:numberOfEmployees), sorted by descending number of employees first, and then alphabetically by company name. It is okay if "soft" only appears as part of the suffix "(software)" in the name.
   d) Retrieve the name and count of software titles produced (see dbo:developer) for all companies that have produced at least 30 such titles. Sort your results in descending order of number of titles.
2. *[10 pts.]* Write a SPARQL construct query that generates all triples inferred by RDFS entailment rule rdfs3 (see the RDF Semantics recommendation [https://www.w3.org/TR/rdf11-mt/#rdfs-entailment], Section 9.2.1). The query should work correctly regardless of domain schema used (i.e., it should be domain independent).
3. *[10 pts.]* Consider a triple store that contains social networking information in the form of FOAF profiles (http://www.foaf-project.org/) collected from various web sites. Assume each pay-level domain uses different URIs for people, but that some sites have users in common. One way to generate links for FOAF is when two resources share the same value for the foaf:mbox property, which provides an e-mail address for an agent. Write a SPARQL construct query to generate owl:sameAs statements based on this property. Make sure that your query only generates triples where the subject and object are different (i.e., non-trivial owl:sameAs statements).
4. *[10 pts.]* Think about the relationship between people and e-mail addresses. What are the pros and cons of generating links as in #3 above? In your answer, consider both precision (what percentage of the generated links are correct) and recall (what percentage of the needed links are generated by the method). Note, I'm not asking you to give percentages; only to say whether the recall and precision of this method will be low or high and why.

Electronic Exercise:

The following exercise requires you to write a Java program using Jena 3.14.0. The Jena distribution includes a number of JAR files, and many (but not all) of these will need to be in your classpath for your program to compile and run. You will need to import classes from the **org.apache.jena.rdf.model** and **org.apache.jena.rdf.query** packages. All of your classes should be placed in a Java package named with your Lehigh LTS account user name (e.g., aaa000), referred to as *userId.* . in the descriptions below. As with the last homework, you must use Jena to parse and query the input files.

5. *[50 pts.]* In this exercise, you will write a program that will create a Web page that groups the publications from ISWC 2019 by country. You will use the actual metadata from the conference (https://iswc2019.semanticweb.org/metadata/), but that data does not have information about the countries of authors. Fortunately for you, I have provided two files, **iswc-org-country.ttl** and **countries.ttl** that provide the additional information you'll need.

   Using Jena, write a class **IswcSummarizer** that can read in all the files in a specified directory and create a Web page that lists all of the publications from three specific conference tracks organized by country. The tracks of interest are "Research," "In-Use" and "Resources." You must read the **iswc-2019-complete.ttl** file plus the two other files into a single Jena model and use a single SPARQL query to retrieve (almost all) the required information from the model. From the command-line, your program should run as:

   ```
   java -cp jenaLoc/lib/*;. userId.IswcSummarizer input-directory output-filename
   ```

   The program should read in the needed files from *input-directory* into a single Jena model. The program will create an HTML file called *output-filename* that has a list of papers grouped by country and sorted by title. Your web page does not have to be fancy, but at a minimum it should contain a head and a body, a title (in the head section), an <h1> heading that describes the page, and second-level (i.e., <h2>) headings for each country. Countries should appear as English names, not as URIs and be in alphabetical order. The papers from each country should be in alphabetical order by title. If a paper has authors from different countries, then it should appear under the heading of each of those countries. However, no paper should appear under a single heading more than once (i.e., no duplicates within the groups). There should only be headings for countries that are the source of at least one paper.

   Each paper should start a new line in the rendered HTML and be preceded with a "(RE)", "(IU)", or "(RS)" string, depending on whether the paper was published in the "Research", "In-use", or "Resources" track, respectively. This should be followed by the title of the paper in double-quotes, the word "by" and finally the correct sequence of the authors (comma-separated, with the word "and" before the last author).

   As I mentioned above, the ISWC 2019 metadata does not provide the countries of papers or authors, but it does provide the affiliations of the authors. The file **iswc-org-country.ttl** provides a mapping from these affiliations to DBPedia country resources. The file **countries.ttl** is an extract from DBPedia containing triples that give the names of each country. Note, many of the countries in this file are historical, and for some strange reason, some resources refer to things that are not countries. Do not worry about this.

   Important: for research papers, author order is significant, and to preserve it, the ISWC metadata stores author information in a somewhat complicated way. Due to this, author information is the only thing you don't have to retrieve in your initial query: Once you have the URI for a paper you may use a secondary process to retrieve the authors in correct order. Each paper has a conf:hasAuthorList property, and this in turn has a node with a conf:hasFirstItem property. Each list item node has two properties: conf:hasContent and conf:next. The object of conf:hasContent will be the URI of an author. The object of conf:next will be a list item node (with its own conf:hasContent and conf:next properties) for the next author. Although it is possible to write a SPARQL query to retrieve the correct order of authors for a paper (using property paths or a special Jena extension to SPARQL), you are allowed to navigate the model using methods from the org.apache.jena.rdf.model package to collect this information. Please note, that you cannot rely on the foaf:made or dc:creator properties to get authors in the correct order!

Submission:

This exercise requires both submission of files and hardcopies of these files. Create a zip file *userId*-prog.zip that contains your source code (.java) files, organized according to Java's file structure (i.e., classes in packages are in subdirectories corresponding to the package naming structure. Do not include any .class or Jena files in your submission, and do not put the code in a "src" directory. The contents of your zip file should be something like:

/*userId*/IswcSummarizer.java
…

Note, I will unzip your file directly into the working directory I will use. I will then run the command as described above. Thus it is important the your submission be organized exactly as I have described. I strongly recommend that before submitting, you unzip your files in a fresh directory, copy the input files, and try to run them using the commands I give above. Once you are confident your zip file will work as expected, submit it via CourseSite.

Print out your .java file(s) and turn in the hardcopy with the rest of your written answers.

Grading:

Your programs will be graded on functionality (90%) and style (10%). Style includes modularity (avoid repeated code when possible, keep methods under ~60 lines, use multiple classes when appropriate), commenting (all of your files should be reasonably commented, including an initial comment that identifies you as the author and descriptive comments for each class and method), proper indentation, clear names, and use of standard naming conventions. The program should also be robust to errors.

If I cannot immediately compile your program or run it using the procedure above, it will be returned to you to fix, and then a late penalty will be assessed depending on how long it takes you to fix it.