Homework #2: SPARQL and Linked Data

The following exercises are due at the beginning of class on Tuesday, Sep. 30. There are two sections: written exercises and electronic exercises. 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 operating on a model using the **swpub.rdf** schema from the last homework. Write the following queries in SPARQL syntax:
   a) Retrieve the titles of all papers that are "articles in a periodical."
   b) For all papers written between 2001 and 2005, inclusive, retrieve the title, human-friendly name (i.e., rdfs:label) of the type (e.g., article, book chapter, etc.) and year of the paper.
   c) Find all papers that appear in a venue with "ISWC" in the name (using partial string matching), and return a list of title, venue, year and topic label (i.e., not the URI of the topic), sorted by year in descending order, followed by topic in ascending order (for those papers in the same year).
   d) For each author, retrieve their name and a count of the number of papers they have authored. Sort the result by author name. You should assume that two authors are the same if and only if their full names match exactly. For sorting purposes, just use the name as it appears, do not worry about first name or last name distinctions. Hint: You need to use specific features of SPARQL 1.1 to do this.
2. *[10 pts.]* Write a SPARQL construct query that generates all triples inferred by RDFS entailment rule rdfs3 (see the RDF Semantics recommendation [http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/], 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.]* Based on your personal experience with how the Internet and Web work, 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).

Electronic Exercise:

The following exercise requires you to write a Java program using Jena 2.12.0. 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 **com.hp.hpl.jena.rdf.model** and **com.hp.hpl.jena.rdf.query** packages. I have provided a revised **SwPub.java** file that is in the **cse428** package for you to use as well; this includes constants for the various classes, properties and other resources defined in **swpub.rdf**. All of your classes should be placed in a Java package named with your Lehigh user id (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.]* Using Jena, write a class **MakeReadListHTML** that can read in all the files in a specified directory and create a Web page that lists all of the publications organized by topic. You must read all of the files into a single model and only use SPARQL to retrieve information from the model. From the command-line, your program should run as:

   `java` *`userId`*`.MakeReadListHTML` *input-directory output-filename*

   The program should read in each file in *input-directory* that has a .rdf suffix into a single Jena model. All required information about the papers must be retrieved using one or more SPARQL queries. The program will create an HTML file called *output-filename* that has a list of papers grouped by topic and sorted by year in descending order. Topics should be output as second-level headings (i.e., with <h2> tags).

   The paper should be output as a complete bibliographic description, using all available information in the file. You need to be able to output descriptions of papers of any of the types defined in **swpub.rdf**. Make sure to output papers that are missing optional properties, such as location, pages, publishedBy, publisherLoc, and publishedMonth.

   If the paper has an electronic version, then the title should be hyperlinked to it (using an <a href="..."> tag). Make sure that the topic labels appear (instead of the topic URIs) and that the authors are listed in correct order (note, you may assume that each paper will have less than 10 authors, which will make this easier).

   Print out your .java file(s) and turn in the hardcopy with the rest of your written answers. Create a zip file *your-user-id*-hw2.zip that contains both your source code (.java) and compiled (.class) files (but do not include any of the Jena files in it). Attach the zip file to the e-mail mentioned above.

Submission:

This exercise requires both submission of files and hardcopies of these files. Print out your .java file(s) and turn in the hardcopy with the rest of your written answers. Submit your electronic files by attaching a zip file to an e-mail sent to **heflin@cse.lehigh.edu** with subject line "CSE 428 – Homework #2 Submission".

Your zip file should be named *userId*-prog.zip and contain 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. If you use the **cse428.SwPub.java** file as is, you do not need to include it, as I will make sure this file is in the classpath when I run your program. Note, I will unzip your file directly into the working directory I will use. I will then run the commands as described for each exercise above. Thus it is important the your submission be organized exactly as I have described. The contents of your zip file should be something like:

/*userId*/MakeReadListHTML.java
…

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.