Proceedings of the 8th International Oracle Life Sciences User Group Meeting 2007
(electronic proceedings: http://www.olsug.org/wiki/images/d/d1/Kelly_Zheng.pdf)

Experience in Integrating Large RDF-based Biomedical Knowledge Resources with Oracle

Kelly Zeng and Olivier Bodenreider
National Library of Medicine, NIH, Bethesda, MD, USA

Introduction and background. Biomedical research often requires the integration of multiple information sources. For example, biological and clinical resources are integrated to support translational research typically. Such information sources are generally large, heterogeneous and not directly machine-processable. We propose to use the Resource Description Framework (RDF) for representing biomedical knowledge and to integrate into Oracle the large quantities of RDF triples resulting from the conversion of existing knowledge bases. This work is a pilot contribution to the *Biomedical Knowledge Repository* under development at the U.S National Library of Medicine as part of the *Advanced Library Services* project.

Acquiring knowledge sources. The two knowledge sources under investigation in this study are Entrez Gene (EG) and the Gene Ontology (GO). While GO is directly available in RDF, we had to convert EG from its original XML representation. An eXtensible Stylesheet Language Transformation (XSLT) approach was used for this purpose. In order to extract a manageable sample, we excluded obsolete records from EG (keeping all "live" genes) and we excluded the less reliable functional annotations of genes (identified by the evidence code "IEA"). The resulting RDF file is 2.2Gb in size.

Integrating knowledge sources into a single Oracle store. The version of Oracle used in this experiment is 10g release 2 with part of the 10.2.0.3 patch (an updated sdordf.jar), on a machine with 2 Xeon CPUs (3.2GHz) running Red Hat Enterprise Linux 4 (RHEL4). The RDF file was converted to NTriple format (1.5G) using the Jena toolkit. The Java batch loader (TestNTriple2NDMBatch) provided by Oracle was used to load the 9.5 millions triples into the database, which took 25.4 hours. The 293,798 triples from the GO in RDF file were converted to NTriple and loaded into the database using the incremental Java loader (TestNTriple2NDM), which took 4.1 hours. Creating indices on the subjects, predicates and objects for a total of 9.8 million triples took 2.3 hours. Finally, we created a rule base (on the 293,798 GO triples) and inserted rules implementing the reflexivity and transitivity of is_a. Building the rule index took 56 minutes.

Querying the RDF store. We used the integrated knowledge store to support biomedical queries, specifically to identify paths between genotype information (e.g., genes annotated with *glycosyltransferase*) and phenotype information (e.g., the disease *congenital muscular dystrophy*). Our typical SPARQL query takes 19 seconds.

Useful practices and lessons learned. The most important finding of this feasibility study is that a moderately large RDF knowledge base (10M triples) was relatively easily manageable, encouraging us to integrate additional resources. From a technical perspective, we encountered the following issues, for which a solution was provided by the Oracle support team. (1) NULL literals and literals with leading blanks caused a loading error. We had to pre-process the files to eliminate those, but Oracle assured us this issue will be addressed in future releases. (2) In a related experiment involving a larger dataset, the datafile exceeded 32Gb (the limit on datafiles with tablespaces in Oracle), resulting in loading failure. To avoid this problem, the "BIGFILE" option must be used (or several smaller datafiles created for the tablespace). (3) The Java incremental loader is about five times slower than the batch loader (103.8 vs. 19.9 triples/second). However, the performance of both loaders is expected to improve in the upcoming 11g release. (4)

Finally, if the RDF file to be loaded contains blank nodes (anonymous nodes), the model_id parameter must be added at the end of the argument string for the Java batch/incremental loader. This is expected to become the default in future releases.