D2K Master
Université Paris Saclay

Information Integration
F. Saïs (sais@lri.fr)

Practical session (3): Jena TDB

Objective: The objective of this practical session is to use the Jena API and the TDB triple store needed when the datasets are very big and cannot fit in memory.

To know have an introduction on RDF and Jena API, you can see the tutorial: https://jena.apache.org/tutorials/rdf_api.html

The needed materials are available at:

https://www.lri.fr/~sais/D2K/RDF/JenaTDB

Jena TDB for RDF

TDB store can be used through Jena API in command line or in a Java application. Here, we will use TDB via a Java application and Eclipse.

Download the file apache-jena-3.1.0.zip and unzip it in your java workspace directory.

In the sub-directory ./lib you will find all the jar files that can add to your java application that uses Jena and TDB.

In the sub-directory ./src-examples you will find a series of examples in Java for the Jena API including the use of TDB.

The Java imports that you may need are:

```
import java.io.InputStream;
import java.util.*;
import org.apache.jena.graph.Graph;
import org.apache.jena.graph.GraphListener;
import org.apache.jena.graph.Node;
import org.apache.jena.graph.NodeFactory;
import org.apache.jena.graph.Triple;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.ResourceFactory;
import org.apache.jena.sparql.core.Quad;
```

```java
import org.apache.jena.sparql.util.graph.GraphListenerBase;
import org.apache.jena.tdb.TDBFactory;
import org.apache.jena.tdb.TDBLoader;
import org.apache.jena.tdb.store.DatasetGraphTDB;
import org.apache.jena.tdb.sys.TDBInternal;
import org.apache.jena.util.FileManager;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.rdf.model.*;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.ReadWrite;
 import org.apache.jena.tdb.base.file.Location;
```

For querying part :

```java
import org.apache.jena.query.Dataset;
import org.apache.jena.query.ReadWrite;
import org.apache.jena.query.Query ;
import org.apache.jena.query.QueryExecution ;
import org.apache.jena.query.QueryExecutionFactory ;
import org.apache.jena.query.QueryFactory ;
import org.apache.jena.query.QuerySolution ;
import org.apache.jena.query.ResultSet ;
```

1) This function allows you create an instance of the dataset given in `file` and put it in the `directory`. This function is executed only one time.

```java
public Dataset createDataset(String file, String directory)
{
	try{
	Dataset dataset = TDBFactory.createDataset(directory);
	dataset.begin(ReadWrite.WRITE) ;
	Model model = dataset.getDefaultModel();
	TDBLoader.loadModel(model, file);
	dataset.commit();
	dataset.end();
	return dataset;
	}catch(Exception ex)
	{
		System.out.println("##### Error Fonction: createDataset #####");
		System.out.println(ex.getMessage());
		return null;
	}

}
```

---

2) One can use the two following instructions to create a Jena model from the dataset created using TDB and stored in `directory.`

```
Dataset d = TDBFactory.createDataset(directory);

Model model = d.getDefaultModel();
```

3) Then, to read the content of the model, you can for example display all the named graphs of the dataset as follows:

```
d.begin(ReadWrite.READ);
try {
      Iterator<Quad> iter = d.asDatasetGraph().find();
      int i=0;
      System.out.println("begin ");
      while (iter.hasNext() && i < 20) {
            Quad quad = iter.next();
            System.out.println("iteration "+i);
            System.out.println(quad);
            i++;
        }
      } finally { d.end(); }
      d.close();
      System.out.println("finish ...");
```

---

4) Query the dataset using SPARQL queries :

a- a simple counting query

```
String sparqlQueryString = "SELECT (count(*) AS ?count) { ?s ?p ?o }" ;
// See http://incubator.apache.org/jena/documentation/query/app_api.html
Query query = QueryFactory.create(sparqlQueryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, d) ;
try {
      ResultSet results = qexec.execSelect() ;
      for ( ; results.hasNext() ; )
      {
            QuerySolution soln = results.nextSolution() ;
            int count = soln.getLiteral("count").getInt() ;
            System.out.println("count = "+count) ;
      }
      } finally { qexec.close() ; }
      // Close the dataset.
      d.close();
```

b - a more complex query with a set of results

```
String sparqlQueryString ="PREFIX humans:
<http://www.inria.fr/2007/09/11/humans.rdfs#> PREFIX inst:
<http://www.inria.fr/2007/09/11/humans.rdfs-instances#> SELECT ?x WHERE {?x
humans:hasSpouse inst:Catherine .}";
// See http://incubator.apache.org/jena/documentation/query/app_api.html

Query query = QueryFactory.create(sparqlQueryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, d) ;
try {
     ResultSet results = qexec.execSelect() ;
     while (results.hasNext()) {
     QuerySolution sol = results.next();
     System.out.println("Solution := "+sol);
     for (Iterator<String> names = sol.varNames(); names.hasNext(); ) {
          final String name = names.next();
          System.out.println("\t"+name+" := "+sol.get(name));
       }
      }
     } finally { qexec.close() ; }

     // Close the dataset.
     d.close();
```

5) Practice SPARQL querying using the queries of the PS the dataset using SPARQL queries of queries-PS3 at :

https://www.lri.fr/~sais/D2K/RDF/corese/queries.zip