GETTING STARTED WITH ONTOTEXT GRAPHDB AND RDF4J

HENRIETTE HARMSE

In this post I will explain how to quickly get started with the free version of Ontotext GraphDB and RDF4J. Ontotext GraphDB is an RDF datastore and RDF4J is a Java framework for accessing RDF datastores (not just GraphDB). I will explain

1. how to install and start GraphDB, as well as how to use the workbench to add a repository, and
2. how to do SPARQL queries against GraphDB using RDF4J.

1. Install and start GraphDB and create a Repository

To gain access to the free version of GraphDB you have to email Ontotext. They will respond with an email with links to a desktop and stand-alone server version of GraphDB. You want to download the stand-alone server version. This is a `graphdb-free-VERSION-dist.zip` file, that you can extract somewhere on your filesystem, which I will refer to here as `$GRAPHDB_ROOT`. To start GraphDB, go to `$GRAPHDB_ROOT/bin` and run `./graphdb`.

To access the workbench you can go to `http://localhost:7200`. To create a new repository, in the left-hand side menu navigate to `Setup-->Repositories`. Click the `Create new repository` button. For our simple example we will use `PersonData` as an `Repository ID`. The rest of the settings we leave as-is. At the bottom of the page you can press the `Create` button.

2. Accessing a GraphDB Repository using RDF4J

To access our `PersonData` repository we will use RDF4J. Since GraphDB is based on the RDF4J libraries, we only need to include the GraphDB dependencies since these already include RDF4J. Thus, in our `pom.xml` file we only need to add the following:

```
<dependency>
    <groupId>com.ontotext.graphdb</groupId>
    <artifactId>graphdb-free-runtime</artifactId>
    <version>8.5.0</version>
</dependency>
```

In our example Java code we first insert some data and then do a query based on the added data. For inserting data we start a transaction and commit it, or, if it fails we do a rollback. For querying the data we iterate through the `TupleQueryResult`, retrieving values for the binding variables we are interested in (i.e. `name` in this case). In line with the `TupleQueryResult` documentation, we close the `TupleQueryResult` once we are done.

Date: 29th June 2018.

```
package org.graphdb.rdf4j.tutorial;

import org.eclipse.rdf4j.model.impl.SimpleLiteral;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryEvaluationException;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.Update;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.repository.http.HTTPRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.Marker;
import org.slf4j.MarkerFactory;

public class SimpleInsertQueryExample {
  private static Logger logger = LoggerFactory.getLogger(SimpleInsertQueryExample.class);
  // Why This Failure marker
  private static final Marker WTF_MARKER = MarkerFactory.getMarker("WTF");

  // GraphDB
  private static final String GRAPHDB_SERVER = "http://localhost:7200/";
  private static final String REPOSITORY_ID = "PersonData";


  private static String strInsert;
  private static String strQuery;

  static {
    strInsert =
        "INSERT DATA {"
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://dbpedia.org/ontology/birthDate> \"1906-1
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://dbpedia.org/ontology/birthPlace> <http:/
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://dbpedia.org/ontology/deathDate> \"1992-0
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://dbpedia.org/ontology/deathPlace> <http:/
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://purl.org/dc/terms/description> \"America
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://xmlns.com/foaf/0.1/gender> \"female\" ."
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://xmlns.com/foaf/0.1/givenName> \"Grace\"
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://xmlns.com/foaf/0.1/name> \"Grace Hopper\
         + "<http://dbpedia.org/resource/Grace_Hopper> <http://xmlns.com/foaf/0.1/surname> \"Hopper\" .
         + "}";

    strQuery =
        "SELECT ?name FROM DEFAULT WHERE {" +
        "?s <http://xmlns.com/foaf/0.1/name> ?name .}";
  }

  private static RepositoryConnection getRepositoryConnection() {
    Repository repository = new HTTPRepository(GRAPHDB_SERVER, REPOSITORY_ID);
    repository.initialize();
```

```
    RepositoryConnection repositoryConnection = repository.getConnection();
    return repositoryConnection;
  }

  private static void insert(RepositoryConnection repositoryConnection) {
    repositoryConnection.begin();

    Update updateOperation = repositoryConnection.prepareUpdate(QueryLanguage.SPARQL, strInsert);
    updateOperation.execute();

    try {
      repositoryConnection.commit();
    } catch (Exception e) {
      if (repositoryConnection.isActive())
        repositoryConnection.rollback();
    }
  }

  private static void query(RepositoryConnection repositoryConnection) {
    TupleQuery tupleQuery = repositoryConnection.prepareTupleQuery(QueryLanguage.SPARQL, strQuery);
    TupleQueryResult result = null;
    try {
      result = tupleQuery.evaluate();
      while (result.hasNext()) {
        BindingSet bindingSet = result.next();

        SimpleLiteral name = (SimpleLiteral)bindingSet.getValue("name");
        logger.trace("name = " + name.stringValue());
      }
    }
    catch (QueryEvaluationException qee) {
      logger.error(WTF_MARKER, qee.getStackTrace().toString(), qee);
    } finally {
      result.close();
    }
  }

  public static void main(String[] args) {
    RepositoryConnection repositoryConnection = null;
    try {
      repositoryConnection = getRepositoryConnection();

      insert(repositoryConnection);
      query(repositoryConnection);

    } catch (Throwable t) {
      logger.error(WTF_MARKER, t.getMessage(), t);
    } finally {
      repositoryConnection.close();
    }
  }
}
```

3. Conclusion

In this brief post I gave a quick example of how you can setup a simple GraphDB repository and query it using SPARQL. You can find sample code on github.