RDF Graph

for
Oracle NoSQL Database

12c Release 1

Library Version DRAFT

ORACLE
NOSQL DATABASE

Legal Notice

Published 3/27/2014

Table of Contents

Preface

This document describes RDF Graph for Oracle NoSQL Database. Both installation and usage of RDF Graph is described in this book. This book assumes you have a working knowledge of both RDF and Oracle NoSQL Database.

This book is intended for application developers using W3C SPARQL endpoint web services and developing Java applications using open source Apache Jena APIs to load, query and inference RDF graph data stored in Oracle NoSQL Database Enterprise Edition, and for the systems engineer responsible for installing RDF Graph for Oracle NoSQL Database.

Conventions Used in This Book

The following typographical conventions are used within in this manual:

Variable or non-literal text is presented in *italics*. For example: "Go to your *KVHOME* directory."

Program examples are displayed in a `monospaced font` on a shaded background. For example:

```
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example16b
{

  public static void main(String[] args) throws Exception
  {

    String szStoreName  = args[0];
    String szHostName   = args[1];
    String szHostPort   = args[2];
    double iSampRate = Double.parseDouble(args[3]);
```

In some situations, programming examples are updated from one chapter to the next. When this occurs, the new code is presented in **`monospaced bold`** font. For example:

```
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example16b
{

  public static void main(String[] args) throws Exception
  {

    String szStoreName  = args[0];
    String szHostName   = args[1];
    String szHostPort   = args[2];
```

```
double iSampRate = Double.parseDouble(args[3]);

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
        = OracleNoSqlConnection.createInstance(szStoreName,
                                               szHostName,
                                               szHostPort);
```

Note

Finally, notes of special interest are represented using a note block such as this.

Chapter 1. RDF Graph for NoSQL Database Overview

This chapter describes the support in Oracle NoSQL Database for semantic technologies, specifically the Worldwide Web Consortium (W3C) Resource Description Framework (RDF), SPARQL query language, and a subset of the Web Ontology Language (OWL). These capabilities are a feature referred to as the *RDF Graph feature for Oracle NoSQL Database Enterprise Edition*.

This chapter assumes that you are familiar with the major concepts associated with RDF and OWL, such as `{subject, predicate, object}` triples, URIs, blank nodes, plain and typed literals, and ontologies. It also assumes that you are familiar with the overall capabilities and use of the Apache Jena Java framework. This chapter does not explain these concepts in detail, but focuses instead on how the concepts are implemented in Oracle NoSQL Database.

For an excellent explanation of RDF concepts, see the World Wide Web Consortium (W3C) RDF Primer at http://www.w3.org/TR/rdf-primer/.

For information about OWL, see the OWL Web Ontology Language Reference at http://www.w3.org/TR/owl-ref/.

The RDF Graph feature provides a Java-based interface to store and query semantic data in the Oracle NoSQL Database. This is done by implementing the well-known Apache Jena Graph, Model, and DatasetGraph APIs. (Apache Jena is an open source framework. For license and copyright conditions, see http://www.apache.org/licences.

The Apache Jena Graph and Model APIs are used to manage graph data, also referred as triples. The DatasetGraph APIs are used to manage named graph data, also referred to as quads.

The RDF Graph feature supports creating a SPARQL end point web service using Apache Jena Joseki, an open source SPARQL server that supports the SPARQL protocol and SPARQL queries. Apache Jena, Apache Jena ARQ and Apache Jena Joseki are included with the RDF Graph feature as described in Prerequisite Software (page 106).

For information about the Apache Jena framework, see http://jena.apache.org/, especially the Apache Jena Documentation page.

Introduction to the RDF Graph Feature

Oracle NoSQL Database enables you to store semantic data and ontologies, to query semantic data and to use inferencing to expand the power of querying on semantic data. The following figure shows how these capabilities interact.

The following illustration shows the relationship between NoSQL Database and the RDF Graph client running in a Web server:

As shown in the previous figure, the database contains semantic data and ontologies (RDF/OWL models), as well as traditional key-value data. To load RDF data, bulk loading is the most efficient approach, although you can load data incrementally using an Apache Jena API.

You can query semantic data and ontologies. You can expand the power of queries on semantic data by using inferencing, which uses rules in rulebases. Inferencing enables you to make logical deductions based on the data and the rules. For information about using rules and rulebases for inferencing, see Inference on an RDF Graph (page 31).

Semantic Data Modeling

In addition to its formal semantics, semantic data has a simple data structure that is effectively modeled using a directed graph. The metadata statements are represented as triples: nodes are used to represent two parts of the triple, and the third part is represented by a directed link that describes the relationship between the nodes.

Statements are expressed in triples: `{subject or resource, predicate or property, object or value}`. In this manual, `{subject, property, object}` is used to describe a triple, and the terms statement and triple may sometimes be used interchangeably. Each triple is a complete and unique fact about a specific domain, and can be represented by a link in a directed graph.

Named Graphs

The RDF Graph feature supports the use of named graphs, which are described in the "RDF Dataset" section of the W3C SPARQL Query Language for RDF recommendation (http://www.w3.org/TR/rdf-sparql-query/#rdfDataset).

This support is provided by extending an RDF triple consisting of the traditional subject, predicate, and object so as to include an additional component representing a graph name. The extended RDF triple, despite having four components, will continue to be referred to as an RDF triple in this document. In addition, the following terms are sometimes used:

- `N-Triple` is a format that does not allow extended triples. Thus, n-triples can include only triples with three components.
- `N-Quad` is a format that allows both "regular" triples (three components) and extended triples (four components, including the graph name). For more information, see N-Quads: Extending N-Triples with Context.

- The graph name component of an RDF triple must either be null or a URI. If it is null, the RDF triple is said to belong to the default graph; otherwise it is said to belong to the named graph whose name is designated by the URI.

Semantic Data in the Database

We store RDF graph data (triples and quads) as key-value pairs in Oracle NoSQL Database. To allow an easy separation of the key-value pairs for RDF graph data, we add a prefix (customizable) to all key-value pairs that are inserted into the Oracle NoSQL Database for RDF related data.

Note that duplicate entries are removed from the store. Duplicate triples and quads are not stored. However, blank nodes are supported because they are part of the RDF modeling.

Each triple (quad) is modeled as a set of K/V pairs in NoSQL Database.

Each graph is managed as a set of quads and stored and encoded as key-value pairs in the Oracle NoSQL Database. Those key-value pairs are used to answer SPARQL queries. In particular, each SPARQL query pattern is translated into a multi-get API call against the Oracle NoSQL Database and the resulting key-value pairs are combined with those from other query patterns to form a final result set for the client.

The following requirements apply to the specifications of URIs and the storage of semantic data in the database:

- A subject must be a URI or a blank node.
- A property must be a URI.
- An object can be any type, such as a URI, a blank node, or a literal. (However, null values and null strings are not supported.)

Loading

To load RDF data into a graph, use one or more of the following options as explained in Load an RDF Graph (page 17):

- Bulk load data from an RDF file with each row containing the three components — subject, predicate, and object — of an RDF triple and optionally a named graph. To load a file containing extended triples (possibly mixed with regular triples) into an Oracle NoSQL Database, the input file must be in N-Quad or TriG format.

  ## Note

  Bulk loading can be performed serially or in parallel.

- Insert triples incrementally.

Inferencing

Inferencing is the ability to make logical deductions based on rules. Inferencing enables you to construct queries that perform semantic matching based on meaningful relationships

among pieces of data, as opposed to just syntactic matching based on string or other values. Inferencing involves the use of rules.

An ontology is a shared conceptualization of knowledge in a particular domain. It consists of a collection of classes, properties, and optionally instances. Classes are typically related by class hierarchy (subclass/superclass relationship). Similarly, the properties can be related by property hierarchy (subproperty/superproperty relationship). Properties can be symmetric or transitive, or both. Properties can also have domain, ranges, and cardinality constraints specified for them.

RDFS-based ontologies only allow specification of class hierarchies, property hierarchies, instanceOf relationships, and a domain and a range for properties.

OWL ontologies build on RDFS-based ontologies by additionally allowing specification of property characteristics. OWL ontologies can be further classified as:

- OWL-Lite

  OWL-Lite restricts the cardinality minimum and maximum values to 0 or 1.

- OWL-DL

  OWL-DL relaxes this restriction by allowing minimum and maximum values.

- OWL Full
  OWL Full allows instances to be also defined as a class, which is not allowed in OWL-DL and OWL-Lite ontologies.

Inferencing with the RDF Graph Feature

The RDF Graph feature supports W3C RDF Schema (RDFS) and Web Ontology Language (OWL) inference through Apache Jena. It also has the ability to support other memory-based third party reasoners such as Pellet and TrOWL as described in Query RDF Graphs (page 19).

Inferencing can be performed in memory on OntModel APIs. Alternatively, the ontology can be passed to an external reasoner. The newly inferred triples can be written back to NoSQL Database and stored as part of the original graph or a new named graph.

Chapter 2. Setup RDF Graph

Setup the System Environment

To use the RDF Graph feature, you must first ensure that the system environment has the necessary software. Please refer to Prerequisite Software (page 106) for the prerequisite software list.

1. Download the RDF Graph feature (rdf_graph_for_nosql_database.zip) from Oracle's Software Delivery Cloud, and unzip it into a temporary directory, such as (on a Linux system) `/tmp/jena_adapter`. (If this temporary directory does not already exist, create it before the unzip operation.)

   See Prerequisite Software (page 106) for complete download instructions.

2. The RDF Graph feature directories and files have the following structure:

```
jar/
    jar/sdordfnosqlclient.jar
javadoc/
 javadoc/javadoc.zip
joseki/
    joseki/index.html
 joseki/application.xml
 joseki/update.html
 joseki/xml-to-html.xsl
 joseki/joseki-config.ttl
web/
 web/web.xml
war/
 war/joseki.war
examples/
    examples/Examples1.java
    examples/Examples1b.java
    examples/Examples1c.java
    examples/Examples1d.java
    examples/Examples2.java
    examples/Examples2b.java
    examples/Examples3.java
    examples/Examples4.java
    examples/Examples4b.java
    examples/Examples5.java
    examples/Examples5b.java
    examples/Examples5c.java
    examples/Examples5d.java
    examples/Examples5e.java
    examples/Examples5f.java
    examples/Examples5g.java
    examples/Examples5h.java
```

```
    examples/Examples6.java
    examples/Examples7.java
    examples/Examples8.java
    examples/Examples9.java
    examples/Examples10.java
    examples/Examples11.java
    examples/Examples12.java
    examples/Examples13.java
    examples/Examples14.java
    examples/Examples15.java
    examples/Examples16.java
    examples/Examples16b.java
    examples/Examples17.java
    examples/example.nt
```

3. The structure of directories and file should also include the following lines:

   ```
   examples/family.rdf
   ```

   ```
   sparqlgateway/
   ```

   ```
   sparqlgateway.war
   ```

   Copy kvclient.jar into <Jena_DIR>/lib (Linux) or <Jena_DIR>\lib (Windows). (kvclient.jar is included in $KVHOME/lib or $KVHOME/lib).

4. If the JAVA_HOME environment variable does not already refer to the JDK 1.6 (update 25 or later) installation, define it accordingly. For example:

   ```
   setenv JAVA_HOME /usr/local/packages/jdk16_u25/
   ```

5. If the SPARQL service to support the SPARQL protocol is not set up, set it up as explained in Setup the SPARQL Service (page 6).

Setup the SPARQL Service

Setting up a SPARQL endpoint using the RDF Graph feature involves creating and deploying a Web Application Archive (WAR) file into a server J2EE container. The RDF Graph feature supports Apache Jena Joseki, an open source SPARQL server that supports the SPARQL protocol and SPARQL queries to create this web application archive.

The following sections explain how to set up a SPARQL service using the bundled web application archive in either Apache Tomcat or Oracle WebLogic Server.

The RDF Graph feature's release package includes a bundled web application archive (joseki.war). Details on how to build a web application archive (joseki.war) for a previous release of the Oracle NoSQL Database or for modification purposes can be found in Generating a New SPARQL Service WAR file (page 107).

Deploy joseki.war

The following steps describe how to deploy the prebundled joseki.war into Apache Tomcat or Oracle WebLogic Server.

1. Ensure that you have downloaded and unzipped the RDF Graph release package for the Oracle NoSQL Database, as explained in Setup the System Environment (page 5).
2. Extract the `joseki-config.ttl` file located in the joseki.war using the following commands:

```
cd /tmp/jena_adapter/war
jar xf joseki.war joseki-config.ttl
```

3. Modify Apache Jena Joseki's configuration file (joseki-config.ttl) to specify the store name, host name, and host port for accessing an Oracle NoSQL database. This data will be used by the SPARQL Service endpoint to establish connections to the Oracle NoSQL Database and execute update and query operations. For detailed information about this configuration, Configuring an Oracle NoSQL Database connection in the SPARQL service (page 10).
4. Rebuild joseki.war by updating the `joseki-config.ttl` as follows:

```
jar uf joseki.war joseki-config.ttl
```

5. Deploy the joseki.war into the selected J2EE container.

Use Apache Tomcat

This section describes how to deploy the SPARQL Service endpoint using Apache Tomcat 7.0. Apache Tomcat is an open source web server implementing Java Servlet and JavaServer Pages (JSP) and providing an HTTP web server environment to run Web applications. Further information and support on Apache Tomcat can be found in http://tomcat.apache.org/.

1. Download and install Apache Tomcat 7.0. For details see http://tomcat.apache.org/tomcat-7.0-doc/index.html.
2. Go to the web application directory of Apache Tomcat Server and copy the joseki.war file as follows. This operation will unpack the war file and deploy the web application. (For information about deploying web application in Apache Tomcat, please refer to http://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html.

```
cd $CATALINA_BASE/webapps
cp -rf /tmp/jena_adapter/joseki.war $CATALINA_HOME/webapps
```

3. Verify your deployment by using your Web browser to connect to a URL in the following format (assume that the Web application is deployed at port 8080):

   http://<hostname>:8080/joseki

   You should see a page titled *Oracle NoSQL Database SPARQL Service Endpoint using Joseki*, and the first text box should contain an example SPARQL query.
4. Click `Submit Query`.

You should see a page titled *Oracle NoSQL Database SPARQL Endpoint Query Results*. There may or may not be any results, depending on the underlying semantic model against which the query is executed.

Use Oracle WebLogic Server

This section describes how to deploy the SPARQL Service endpoint using Oracle WebLogic Server 12c. For information about Oracle WebLogic Server please refer to http://www.oracle.com/technology/products/weblogic/.

1. Download and Install Oracle WebLogic Server 12c Release 1 (12.1.1). For details, see http://www.oracle.com/technology/products/weblogic/ and http://www.oracle.com/technetwork/middleware/ias/downloads/wls-main-097127.html.

2. Go to the auto-deploy directory of the WebLogic Server installation and copy files, as follows. (For information about auto-deploying applications in development domains refer to the following document: http://docs.oracle.com/cd/E24329_01/web.1211/e24443/autodeploy.htm)

   ```
   cd <domain_name>/autodeploy
   cp -rf /tmp/joseki.war <domain_name>/autodeploy
   ```

   In the preceding example, <domain_name> is the name of a WebLogic Server domain.

   Note that you can run a WebLogic Server domain in two different modes: development and production. However, only development mode allows you use the auto-deployment feature.

3. Verify your deployment by using your Web browser to connect to a URL in the following format (assume that port 7001 is used):

   http://<hostname>:7001/joseki

   You should see a page titled *Oracle NoSQL Database SPARQL Service Endpoint using Joseki*, and the first text box should contain an example SPARQL query.

4. Click `Submit Query`.

   > **Note**
   > You should see a page titled *Oracle NoSQL Database SPARQL Endpoint Query Results*. There may or may not be any results, depending on the underlying semantic model against which the query is executed.

By default, the `joseki-config.ttl` file contains an oracle-nosql:Dataset definition using all graphs stored in the Oracle NoSQL Database. The following snippet shows the configuration.

```
<#oracle> rdf:type oracle-nosql:Dataset;
joseki:poolSize 3;  ## Number of concurrent connections allowed
                    ## to this dataset.
oracle-nosql:connection  ## NoSQL connection
[
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName "localhost";
oracle-nosql:storeName "mystore";
oracle-nosql:hostPort "5000";
```

```
];
oracle-nosql:allGraphs [] .       ## Graph descriptions
```

The `oracle-nosql:allGraphs` predicate denotes that the SPARQL Service endpoint will serve queries using all named graphs (including the default graph) stored in an Oracle NoSQL Database. You can also specify the rulebase(s) to use when serving queries. In the following example, the `oracle:ruleBaseID` predicate denotes that the SPARQL Service endpoint should serve queries using all named graphs (including asserted and inferred triples marked with the rulebase ID 1.

```
<#oracle> rdf:type oracle-nosql:Dataset;
joseki:poolSize 3;  ## Number of concurrent connections allowed
                    ## to this dataset.
oracle-nosql:connection  ## NoSQL connection
[
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName "localhost";
oracle-nosql:storeName "mystore";
oracle-nosql:hostPort "5000";
];
oracle-nosql:allGraphs [ oracle-nosql:ruleBaseID "1" .  ] .
## Graph descriptions
```

If you require the SPARQL Service endpoint to serve queries using only a specified set of graph names, then use the `oracle-nosql:namedGraph` predicate instead of `oracle-nosql:allGraphs`. Further details can be found in Configuring the SPARQL Service: Oracle NoSQL Database (page 11).

You can add a few example triples and quads to test the named graph functions using the following Java code snippet:

```
public static void main(String[] args) throws Exception
{
String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

System.out.println("Create Oracle NoSQL connection");
OracleNoSqlConnection conn  =
OracleNoSqlConnection.createInstance(szStoreName,
                                     szHostName,
                                     szHostPort);
System.out.println("Create Oracle NoSQL graph and dataset ");
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// add data to the bob named graph
datasetGraph.add(new Quad(
```

```
        Node.createURI("http://example.org/bob"), // graph name
        Node.createURI("urn:bob"),
        Node.createURI("http://xmlns.com/foaf/0.1/name"),
        Node.createLiteral("Bob")));

// add data to the alice named graph
datasetGraph.add(new Quad(
        Node.createURI("http://example.org/alice"), // graph name
        Node.createURI("urn:alice"),
        Node.createURI("http://xmlns.com/foaf/0.1/name"),
        Node.createLiteral("Alice")));
ds.close();
conn.dispose();
}
```

After inserting the quads, go to http://<hostname>:7001/joseki (Oracle WebLogic Server) or http://<hostname>:8080/joseki (Apache Tomcat), type the following SPARQL query, and click `Submit Query`:

```
SELECT ?g ?s ?p ?o
WHERE
{ GRAPH ?g
   { ?s ?p ?o}
}
```

The result should be an HTML table with four columns and two sets of result bindings.

This page also contains a JSON Output option. If this option is selected (enabled), the SPARQL query response is converted to JSON format.

Configuring an Oracle NoSQL Database connection in the SPARQL service

To configure the connections to the Oracle NoSQL Database, you must modify Apache Jena Joseki's configuration file (`joseki-config.ttl`) file located in `/tmp/joseki.war` to include the store name, host name, and host port to access the Oracle NoSQL Database. This data is used by the SPARQL Service endpoint to establish connections to the Oracle NoSQL Database and execute update and query operations.

To define this data, refer to the `oracle-nosql:connection` predicate in the `joseki-config.ttl`. This predicate denotes that the SPARQL service endpoint will connect to an Oracle NoSQL Database named <*store_name*> accessible through the host name <*host_name*> and port <*host_port*>. The following snippet shows the configuration.

```
<#oracle> rdf:type oracle-nosql:Dataset;
joseki:poolSize 1;  ## Number of concurrent connections allowed to
                    ## this data set.
oracle-nosql:connection  ## NoSQL connection
[
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName <host_name>;
oracle-nosql:storeName <store_name>;
```

```
oracle-nosql:hostPort <host_port>;
];
...
```

Configuring the SPARQL Service: Oracle NoSQL Database

By default, the SPARQL Service endpoint assumes that queries are going to be executed against all named graphs (including the default graph) stored in the specified Oracle NoSQL database. Users can configure these settings to serve queries using only the default graph or a subset of named graphs by editing the `joseki-config.ttl` configuration file, which is in `<domain_name>/autodeploy/joseki.war`.

The supplied `joseki-config.ttl` file includes a section similar to the following for the Oracle NoSQL Database data set:

```
#
## Datasets
#
[] ja:loadClass
"oracle.spatial.rdf.client.jena.assembler.OracleAssemblerVocab" .

oracle-nosql:Dataset rdfs:subClassOf ja:RDFDataset .

<#oracle> rdf:type oracle-nosql:Dataset;
## Number of concurrent connections allowed to this dataset.
joseki:poolSize 1;

oracle-nosql:connection  ## connection to an Oracle NoSQL Database
  [
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName "localhost";
    oracle-nosql:storeName "mystore";
       oracle-nosql:hostPort "5000";
];

oracle-nosql:allGraphs [] .
```

In this section of the file, you can:

- Modify the `joseki:poolSize` value, which specifies the number of concurrent connections allowed to this Oracle NoSQL data set (<#oracle> rdf:type oracle-nosql:Dataset;), which points to various RDF models in the Oracle NoSQL Database.
- Specify the default graph used to serve queries using the property `oracle-nosql:defaultGraph` as follows:

```
<#oracle> rdf:type oracle-nosql:Dataset;
joseki:poolSize 1; ## Number of concurrent connections allowed to
                   ## this data set.
oracle-nosql:connection  ## NoSQL connection
[
```

```
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName "localhost";
oracle-nosql:storeName "mystore";
oracle-nosql:hostPort "5000";
];
oracle-nosql:defaultGraph [] .
```

- The `oracle:defaultGraph` predicate denotes that the SPARQL Service endpoint should serve queries using the default graph (consisting of triples that have no or NULL graph names), if stored in the Oracle NoSQL Database.
- If you require the SPARQL service endpoint to serve queries using asserted and inferred triples from a default graph, you should specify the rulebase ID of the inferred triples. The `oracle:ruleBaseID` predicate denotes that the endpoint should include all triples marked with the specified rulebase ID.
- For example, the following specifies rulebase ID 1 for the default graph.

```
oracle-nosql:defaultGraph [
 oracle-nosql:ruleBaseID "1" .
] .
```

- Specify a subset of named graphs that the SPARQL Service endpoint will use to serve queries. For example, you can specify two named graphs called <http://G1> and <http://G2> as follows:

```
<#oracle>> rdf:type oracle-nosql:Dataset;
joseki:poolSize 1; ## Number of concurrent connections allowed to
                   ## this data set.
oracle-nosql:connection  ## NoSQL connection
[
rdf:type oracle-nosql:NoSQLConnection;
oracle-nosql:hostName "localhost";
oracle-nosql:storeName "mystore";
oracle-nosql:hostPort "5000";
];

 oracle-nosql:namedGraph [ oracle-nosql:graphName <http://G1> ] .
 oracle-nosql:namedGraph [ oracle-nosql:graphName <http://G2> ] .
```

- The `oracle-nosql:namedGraph` predicate denotes that the SPARQL Service endpoint should serve queries using the named graph with a graph name denoted by `oracle-nosql:graphName`, if stored in the Oracle NoSQL Database. In this example, the SPARQL Service endpoint will only serve queries using two named graphs <http://G1> and <http://G2>. This way, any triple belonging to a different named graph will not be considered.
- If you require the SPARQL service endpoint to serve queries using asserted and inferred triples of a named graph, you should specify the rulebase ID of the inferred triples. The `oracle:ruleBaseID` predicate denotes that the endpoint should include all triples marked with the specified rulebase ID. For example, the following specifies that a rulebase ID, identified with 1, should be used in conjunction with named graph <http://G1>.

```
oracle-nosql:namedGraph [
 oracle-nosql:graphName <http://G1> ;
    oracle-nosql:ruleBaseID "1" .
] .
```

- Note that when using this configuration, one can still use SPARQL Update requests to create new graphs or add data to named graphs. However, queries against named graphs not specified in this configuration will not return any matches.

Chapter 3. Connect to NoSQL Database

This section describes two ways the RDF Graph feature can connect to Oracle NoSQL Database. For comprehensive documentation of the API calls that support the connections, see the RDF Graph feature reference information (Javadoc).

Making a single connection to an Oracle NoSQL Database

The RDF Graph feature provides a convenient handler to manage connections and operations to the Oracle NoSQL Database. This handler represents a relevant component used by the OracleGraphNoSql and the DatasetGraphNoSql to access and persist all RDF data in the Oracle NoSQL Database.

A connection handler is provided through the RDF Graph feature OracleNoSQLConnection class. Once this class is initialized, you can use this connection object to load, modify, query, and remove RDF triple or quad data from the Oracle NoSQL Database through the Graph and DatasetGraph APIs.

The following example sets up an OracleNoSqlConnection object.

```
public static void main(String[] args) throws Exception
{
String szStoreName = args[0];
String szHostName  = args[1];
String szHostPort  = args[2];
String szModelName = args[3];

System.out.println("Creating connection handler");
OracleNoSqlConnection conn
                      = OracleNoSqlConnection.createInstance(szStoreName,
                                                             szHostName,
                                                             szHostPort);
OracleModelNoSql model
    = OracleModelNoSql.createOracleModelNoSql(szModelName,conn);

System.out.println("Clear model");
model.removeAll();

model.getGraph().add(Triple.create(Node.createURI("u:John"),
                                   Node.createURI("u:cousinOf"),
                                   Node.createURI("u:Jackie")));


String queryString = "select ?x ?y ?z WHERE {?x ?y ?z}";

System.out.println("Execute query " + queryString);

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);
```

```
try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();

// Close connection
conn.dispose();
}
```

Connection Pooling

Oracle NoSQL Database Connection Pooling is provided through the RDF Graph feature OraclePoolNoSql class. Once this class is initialized, it can return OracleNoSqlConnection objects out of its pool of available connections. These objects are essentially wrappers to Oracle NoSQL Database connections. After dispose is called on the OracleNoSqlConnection object, instead of being closed the connection is actually returned to the pool. More information about using OraclePoolNoSql can be found in the API reference information (Javadoc).

The following example sets up an OraclePoolNoSql object with three initial connections.

```
public static void main(String[] args) throws Exception
{
String szStoreName = args[0];
String szHostName = args[1];
String szHostPort = args[2];
String szModelName = args[3];

// Property of the pool: wait if no connection is available at request.
boolean bWaitIfBusy = true;

System.out.println("Creating OracleNoSQL pool");
OracleNoSqlPool pool =
                OracleNoSqlPool.createInstance(szStoreName,
                                               szHostName,
                                               szHostPort,
                                               3, // pool size
                                               bWaitIfBusy,
                                               true);  // lazyInit
System.out.println("Done creating OracleNoSql pool");

// grab an Oracle NoSQL connection and do something
System.out.println("Get a connection from the pool");
```

```
OracleNoSqlConnection conn = pool.getResource();

OracleModelNoSql model =
           OracleModelNoSql.createOracleModelNoSql(szModelName, conn);

System.out.println("Clear model");
model.removeAll();

model.getGraph().add(Triple.create(Node.createURI("u:John"),
                                   Node.createURI("u:cousinOf"),
                                   Node.createURI("u:Jackie")));
model.close();

//return connection back to the pool
conn.dispose();

// grab another Oracle NoSQL connection and do something
System.out.println("Get a connection from the pool");
conn = pool.getResource();

String queryString = "select ?x ?y ?z WHERE {?x ?y ?z}";

System.out.println("Execute query " + queryString);

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();

//return connection back to the pool
    conn.dispose();

// Close pool.
// This will close all resources even if they have not been freed up
   System.out.println("Close pool, this will close all resources");
   pool.close();

}
```

Chapter 4. Load an RDF Graph

The RDF Graph feature supports loading RDF triples data into the default graph or a named graph in Oracle NoSQL Database. RDF data can be loaded into the graph using two approaches: Triples can be inserted incrementally using the `graph.add(Triple.create())` API as illustrated in Example1.java: Create a default graph and add/delete triples (page 35) and Example1b.java: Create a named graph and add/delete triples (page 37).

Triples can be bulk loaded from an RDF file using the `DatasetGraphNoSql.load()` API as illustrated in Example2.java: Load an RDF file (page 44) and Concurrent RDF data loading (page 46).

Parallel Loading Using the RDF Graph feature

To load RDF data files containing thousands to millions of records into an Oracle NoSQL Database, you can use concurrent loading in the RDF Graph feature to speed up the task.

Concurrent or parallel loading is an optimized solution to data loading in the RDF Graph feature, where triples are organized into batches and load execution is done if and only if a batch is full or the process has loaded all triples from the RDF file. Once a batch is full, to increase performance on write operations to Oracle NoSQL Database, we use multiple threads and connections to store multiple triples into the Oracle NoSQL Database.

You can use parallel loading by specifying the degree of parallelism (number of threads used in load operations) and the size of the batches managed when calling the load method in the OracleDatasetGraphNoSql class.

The following example loads an RDF data file in Oracle NoSQL Database using parallel loading. The degree of parallelism and batch size used are controlled by the input parameters iDOP and iBatchSize respectively.

On a balanced hardware setup with 4 or more CPU cores, setting a DOP to 8 (or 16) can improve significantly the speed of the load operation when many triples are going to be processed.

```
public static void main(String[] args) throws Exception
{
String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
int iBatchSize      = Integer.parseInt(args[3]);
int iDOP            = Integer.parseInt(args[4]);

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
= OracleNoSqlConnection.createInstance(szStoreName,
                                       szHostName,
                                       szHostPort);

// Create Oracle NoSQL datasetgraph
```

```
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear datasetgraph
datasetGraph.clearRepository();

// Load N-QUADS data from a file into the Oracle NoSQL Database
DatasetGraphNoSql.load("example.nt",
                       Lang.NQUADS,         // data format
                       conn,
                       "http://example.org",
                       iBatchSize,          // batch size
                       iDOP);               // degree of parallelism

// Create dataset from Oracle NoSQL datasetgraph to execute
Dataset ds = DatasetImpl.wrap(datasetGraph);

String szQuery = "select * where { graph ?g { ?s ?p ?o }  }";
System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
   }
}
```

Chapter 5. Query RDF Graphs

The RDF Graph feature has native support for World Wide Web Consortium (W3C) standards. SPARQL is a query language designed specifically for graph pattern matching queries, and RDF and OWL are standards for representing and defining graph data. Oracle NoSQL Database stores RDF data as an array of bytes. It supports non-ASCII characters to accommodate a wide range of international characters.

Functions Supported in SPARQL Queries

SPARQL queries can use functions in the function library of the Apache Jena ARQ query engine. These queries are executed in the middle tier.

The following examples use the upper-case and namespace functions. In these examples, the prefix *fn* is http://www.w3.org/2005/xpath-functions# and the prefix *afn* is http://jena.hpl.hp.com/ARQ/function#.

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX fn:  <http://www.w3.org/2005/xpath-functions#>
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
SELECT (fn:upper-case(?object) as ?object1)
WHERE { ?subject dc:title ?object }

PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
SELECT ?subject (afn:namespace(?object) as ?object1)
WHERE {
?subject <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
?object }
```

Syntax Involving Bnodes (Blank Nodes)

Syntax involving bnodes can be used freely in query patterns. For example, the following bnode-related syntax is supported at the parser level, so each is equivalent to its full triple-query-pattern-based version.

```
:x :q [ :p "v" ] .
(1 ?x 3 4) :p "w" .
(1 [:p :q] ( 2 ) ) .
```

JavaScript Object Notation (JSON) Format Support

JavaScript Object Notation (JSON) format is supported for SPARQL query responses. JSON data format is simple, compact, and well suited for JavaScript programs.

For example, consider the following Java code snippet, which executes a query over the data stored in the Oracle NoSQL Database, and then calls the `ResultSetFormatter.outputAsJSON()` method to present the retrieved results:

```
OracleNoSqlConnection conn =
```

```
                    OracleNoSqlConnection.createInstance(storeName,
                                                         hostName,
                                                         hostPort);
OracleGraphNoSql graph = new OracleNamedGraphNoSql(graphName, conn);
OracleModelNoSql model = new OracleModelNoSql(graph);

graph.add(new Triple(Node.createURI("http://ds1"),
Node.createURI("http://dp1"),
Node.createURI("http://do1")));

graph.add(new Triple(Node.createURI("http://ds2"),
Node.createURI("http://dp2"),
Node.createURI("http://do2")));

Query q = QueryFactory.create("select ?s ?p ?o where {?s ?p ?o}",
Syntax.syntaxARQ);

QueryExecution qexec = QueryExecutionFactory.create(q, model);
ResultSet results = qexec.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results);
```

After the execution of this code, the following JSON output is produced:

```
{
"head": {
"vars": [ "s" , "p" , "o" ]
} ,
"results": {
"bindings": [
{
"s": { "type": "uri" , "value": "http://ds1" } ,
"p": { "type": "uri" , "value": "http://dp1" } ,
"o": { "type": "uri" , "value": "http://do1" }
} ,
{
"s": { "type": "uri" , "value": "http://ds2" } ,
"p": { "type": "uri" , "value": "http://dp2" } ,
"o": { "type": "uri" , "value": "http://do2" }
}
]
}
}
```

The preceding example can be modified to execute a query over a remote SPARQL endpoint instead of executing it directly against an Oracle NoSQL Database. (If the remote SPARQL endpoint is outside a firewall, then the HTTP Proxy probably needs to be set.)

```
Query q = QueryFactory.create("select ?s ?p ?o where {?s ?p ?o}",
Syntax.syntaxARQ);
QueryExecution qexec =
               QueryExecutionFactory.sparqlService(sparqlURL, q);
```

```
ResultSet results = qexec.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results);
```

To extend the first example in this section to named graphs, the following code snippet adds two quads to the same dataset, executes a named graph-based SPARQL query, and serializes the query output into JSON format:

```
DatasetGraphNoSql dsg = DatasetGraphNoSql.createFrom(graph);
graph.close();

dsg.add(new Quad(Node.createURI("http://g1"),
Node.createURI("http://s1"),
Node.createURI("http://p1"),
Node.createURI("http://o1") )
);

dsg.add(new Quad(Node.createURI("http://g2"),
                 Node.createURI("http://s2"),
                 Node.createURI("http://p2"),
                 Node.createURI("http://o2") )
        );

Query q1 = QueryFactory.create(
                 "select ?g ?s ?p ?o where { GRAPH ?g {?s ?p ?o} }");

QueryExecution qexec1 =
             QueryExecutionFactory.create(q1, DatasetImpl.wrap(dsg));

ResultSet results1 = qexec1.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results1);
dsg.close();
conn.dispose();
```

The JSON output produced after executing the code is presented as follows:

```
{
"head": {
"vars": [ "g" , "s" , "p" , "o" ]
} ,
"results": {
"bindings": [
{
"g": { "type": "uri" , "value": "http://g1" } ,
"s": { "type": "uri" , "value": "http://s1" } ,
"p": { "type": "uri" , "value": "http://p1" } ,
"o": { "type": "uri" , "value": "http://o1" }
} ,
{
"g": { "type": "uri" , "value": "http://g2" } ,
"s": { "type": "uri" , "value": "http://s2" } ,
"p": { "type": "uri" , "value": "http://p2" } ,
```

```
"o": { "type": "uri" , "value": "http://o2" }
}
]
}
}
```

You can also get a JSON response through HTTP against a Joseki-based SPARQL endpoint, as in the following example. Normally, when executing a SPARQL query against a SPARQL Web service endpoint, the Accept request-head field is set to be application/sparql-results+xml. For JSON output format, replace the Accept request-head field with application/sparql-results +json.

```
http://hostname:7001/joseki/oracle-nosql?query=
  <URL_ENCODED_SPARQL_QUERY>&output=json
```

Best Practices

Additional Query Options

The RDF Graph feature allows you to specify additional query options. It implements these capabilities by using the SPARQL namespace prefix syntax to refer to Oracle-specific namespaces that contain these query options. The namespaces are defined in the form PREFIX ORACLE_SEM_FS_NS.

Additional query options can be passed to a SPARQL query by including a line in the following form:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#option>
```

The *option* reflects a query setting (or multiple query options delimited by commas) to be applied to the SPARQL query execution. For example:

```
PREFIX ORACLE_SEM_FS_NS:
<http://oracle.com/semtech#TIMEOUT=3,DOP=4,ORDERED>
SELECT * WHERE {?subject ?property ?object }
```

The following query options are supported:

- `ASSERTED_ONLY` causes only the asserted triples/quads to be queried.
- `BATCH=n` specifies the size of the batches (n) used to execute concurrent retrieval of bindings. Using a batch size that is larger than the default of 1,000, such as 5,000 or 10,000 when retrieving RDF data from the Oracle NoSQL Database may improve performance.
- `BEST_EFFORT_QUERY=T`, when used with the TIMEOUT=n option, returns all matches found in n seconds for the SPARQL query.
- `DOP=n` specifies the degree of parallelism (n) for the query. The default value is 1. With multi-core or multi-CPU processors, experimenting with different DOP values (such as 4 or 8) may improve performance. A good starting point for DOP can be the number of CPU cores, assuming the level of query concurrency is low. To ensure that no single query

dominates the CPU resources, DOP should be set at a lower value when the number of concurrent requests increases.

- `INCLUDE=RULEBASE_ID=n` specifies the rulebase ID to use when answering a SPARQL query. This query option will override any rulebase configuration defined at the SPARQL Service endpoint.
- `INF_ONLY` causes only the inferred triples/quads to be queried.
- `JENA_EXECUTOR` disables the compilation of SPARQL queries to the RDF Graph feature; instead, the Apache Jena native query executor will be used.
- `JOIN_METHOD={nl, hash}` specifies how query patterns in a SPARQL query can be joined, either a nested loop join (nl) or hash join (hash) method can be used. For more information, see JOIN_METHOD option (page 23).
- `ORDERED` specifies that query patterns in a SPARQL query should be executed in the same order as they are specified.
- `TIMEOUT=n (query timeout)` specifies the number of seconds (n) that the query will run until it is terminated. The underlying query execution generated from a SPARQL query can return many matches and can use features like sub-queries and assignments, all of which can take considerable time. The TIMEOUT and BEST_EFFORT_QUERY=t options can be used to prevent what you consider excessive processing time for the query.

JOIN_METHOD option

A SPARQL query consists of a single (or multiple) query patterns, conjunctions, disjunctions, and optional triple patterns. The RDF Graph feature processes triple patterns in the SPARQL query and executes join operations over their partial results to retrieve query results. The RDF Graph feature automatically analyzes the received SPARQL query and determines an execution plan using an efficient join operation between two query row sources (outer and inner, left or right). A query row source consists of a query pattern or the intermediate results from another join operation.

However, you can use the JOIN_METHOD option that uses the RDF Graph feature to specify which join operation to use in SPARQL query execution. For example, assume the following query:

```
PREFIX ORACLE_SEM_FS_NS:<http://oracle.com/semtech#JOIN_METHOD=NL>
SELECT ?subject ?object ?grandkid
WHERE {
?subject <u:parentOf>  ?object   .
?object  <u:parentOf>  ?grandkid .
}
```

In this case, the join method to use will be set to nested loop join. The first (outer) query portion of this query (in this case query pattern ?subject u:parentOf> ?object), is executed against the Oracle NoSQL Database. Each binding of ?object from the results is then pushed into the second (inner) query pattern (in this case ?object <u:parentOf> ?grandkid), and

which in turn is then executed against the Oracle NoSQL Database. Note that nested loop join operations can be executed only if the inner row source is a query pattern.

If the join method to use is set to `hash join`, both the outer row source and inner row source of this query will be executed against the Oracle NoSQL Database. All results from the outer row source (also called the build table) will be stored in a hash table structure with respect to its binding of `?object`, as it is a common variable between the outer and inner row sources. Then, each binding of `?object` from the inner row source (also called the probe table) will be hashed and matched against the hash data structure.

SPARQL 1.1 federated query SERVICE Clause

When writing a SPARQL 1.1 federated query, you can set a limit on returned rows in the sub-query inside the SERVICE clause. This can effectively constrain the amount of data to be transported between the local repository and the remote SPARQL endpoint.

For example, the following query specifies a limit of 100 in the subquery in the SERVICE clause:

```
PREFIX : <http://example.com/>
SELECT ?s ?o
WHERE
{
?s :name "CA"
SERVICE <http://REMOTE_SPARQL_ENDPOINT_HERE>
{
select ?s ?o
{?s :info ?o}
limit 100
}
}
```

Data sampling

Having sufficient statistics for the query optimizer is critical for good query performance. In general, you should ensure that you have gathered basic statistics for the RDF Graph feature to use during query execution. In Oracle NoSQL Database, these statistics are generated by maintaining data sampling.

Data sampling is defined as a representative subset of triples from an RDF graph (or dataset) stored in an Oracle NoSQL Database, generated at a certain point of time. The size of this subset is determined by the size of the overall data and a sampling rate. Data sampling is automatically performed when an RDF data file is loaded into or removed from the Oracle NoSQL Database. By default, the data sampling rate is 0.003 (or 3 per 1000). The default sampling rate may not be adequate for all database sizes. It may improve performance to reduce the sampling rate for substantially larger data sets to retain a more manageable count of sampled data. For instance, performance may be improved by setting the sampling as 0.0001 for billions of triples and 0.00001 for trillions of triples.

Data sampling service is provided through the method analyze RDF Graph feature `OracleGraphNoSql` and `DatasetGraphNoSql` class. This method essentially gets all the data

from the graph (or dataset) and generates a representative subset used as data sampling. Users can choose the size of data sampling by specifying the *samplingRate*. Note that existing data sampling will be removed once this operation is executed. More information about using analyze can be found in the API reference information (Javadoc).

The following example analyzes the data from a graph and generates a sampling subset with a sampling rate of 0.005 (or 5/1000).

```
public static void main(String[] args) throws Exception
{
String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

System.out.println("Create Oracle NoSQL connection");
OracleNoSqlConnection conn
               = OracleNoSqlConnection.createInstance(szStoreName,
                                                      szHostName,
                                                      szHostPort);

System.out.println("Create named graph");
OracleGraphNoSql graph = new OracleGraphNoSql(conn);

System.out.println("Clear graph");
graph.clearRepository();


System.out.println("Load data from file into a NoSQL database");

DatasetGraphNoSql.load("family.rdf", Lang.RDFXML, conn,
                           "http://example.com"); // base URI

System.out.println(“Analyze data”);
long sizeSamp = graph.analyze(0.005); // 5 out of 1000

System.out.println("sampling size is " + sizeSamp);

graph.close();
conn.dispose();
}
```

Query hints

The RDF Graph feature allows you to include query optimization hints in a SPARQL query. It implements these capabilities by using the SPARQL namespace prefix syntax to refer to Oracle-specific namespaces that contain these hints. The namespace is defined in the form PREFIX ORACLE_SEM_HT_NS.

Query hints can be passed to a SPARQL query by including a line in the following form:

```
PREFIX ORACLE_SEM_HT_NS: <http://oracle.com/semtech#hint>
```

Where *hint* reflects any hint supported by the RDF Graph feature.

A query hint represents a helper for the RDF Graph feature to generate an execution plan used to execute a SPARQL query. An execution plan determines the way query patterns will be handled by the RDF Graph feature. This involves the following conditions:

1. The order in which query patterns in a Basic Graph Pattern will be executed.
2. How query patterns will be joined together in order to complete a query execution.
3. The join method (*nested loop join* or *hash join*) to pick in order to merge results retrieved from two query patterns or pre-calculated results.

An *execution plan* is written using post-fix notation. In this notation, joins operations (expressed as HJ or NLJ) are preceded by its operands (the result of another join operation or a query pattern). The order in which the operands in a join operation are presented is relevant to query execution as the number of operations executed in the join operation are intimately related to the size of these operands. This, in consequence will affect the performance of a query execution.

Query patterns in a plan are expressed as QP< ID>, where ID represents the position of the query pattern with respect to the specified SPARQL query. Additionally, every join operation and its respective operands should be wrapped using parentheses.

For example, consider the following SPARQL query that retrieves all pairs of names of people who know each other.

```
PREFIX ORACLE_SEM_HT_NS: <http://oracle.com/semtech#plan=
                                          ((qp2%20qp3%20NLJ)%20qp1%20HJ)>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +

SELECT ?name1 ?name2 " +
WHERE {
graph <http://example.org/graph> {
?person1 foaf:knows ?person2     .         #QP1
?person1 foaf:name ?name1       .         #QP2
?person2 foaf:name ?name2         .        #QP3
}
```

Suppose that we want to specify an execution plan that will perform first a nested loop join operation between `?person1 foaf:name ?name1` and `?person1 foaf:knows ?person2`, and then perform a hash join operation between the results and the third query pattern `?person2 foaf:name ?name2`. This plan can be defined using post-fix notation as follows:

```
(
(
( ?person1 foaf:name ?name1 )
( ?person1 foaf:name ?name2 )
 NLJ )

( ?person1 foaf:knows ?person2 )
HJ )
```

This execution plan can be specified into the RDF Graph feature using the query `hint PLAN=`*`encoded_plan`*, where *encoded_plan* represents an URL encoded representation of an execution plan to execute all the query patterns included in a SPARQL query using hash join or nested loop join operations. Query hints can only be applied to SPARQL queries with a single BGP.

Note that if a plan is not UTF-8 encoded, does not include all query patterns in a SPARQL query, or is syntactically incorrect, this hint will be ignored and the RDF Graph feature will continue with a default query optimization and execution. For information about queries and joins operations, see JOIN_METHOD option (page 23).

Chapter 6. Update an RDF Graph

The RDF Graph feature supports SPARQL Update (http://www.w3.org/TR/sparql11-update/), also referred to as SPARUL. The primary programming APIs involve the Apache Jena class `UpdateAction` (in package `com.hp.hpl.jena.update`) and the RDF Graph feature classes `OracleGraphNoSql` and `DatasetGraphNoSql`. The following example shows a SPARQL Update operation that removes all triples in named graph `<http://example/graph>` from the relevant model stored in the database.

```
OracleGraphNoSql oracleGraph = .... ;
DatasetGraphNoSql dsgos = DatasetGraphNoSql.createFrom(oracleGraph);

// SPARQL Update operation
String szUpdateAction = "DROP GRAPH <http://example/graph>";
// Execute the Update against a DatasetGraph instance
// (can be a Jena Model as well)
UpdateAction.parseExecute(szUpdateAction, dsgos);
```

Note that the Oracle NoSQL Database does not keep any information about an empty named graph. This implies that if you invoke the CREATE GRAPH <graph_name> without adding any triples into this graph, then no triples will be created. With an Oracle NoSQL Database, you can safely skip the CREATE GRAPH step, as is the case in the following example.

```
PREFIX example: <http://example/>
INSERT DATA {
    GRAPH <http://example/graph> {
  example:anne example:age 30 .
  example:peter example:birthyear 1982
 }
} ;
DELETE DATA {
     GRAPH <http://example/graph> { example:anne example:age 30 . }
}
```

The following example shows a SPARQL Update operation (from ARQ 2.9.2) involving multiple insert and delete operations.

```
PREFIX : <http://example/>
CREATE GRAPH <http://example/graph>;
INSERT DATA { example:anne example:age 30 };
INSERT DATA { example:peter example:birthyear 1982 };
DELETE DATA { example:peter example:birthyear 1982 };
INSERT DATA {
    GRAPH <http://example/graph> {
  example:anne example:age 30 .
  example:peter example:birthyear 1982
 }
};
DELETE DATA {
     GRAPH <http://example/graph> { example:anne example:age 30 }
}
```

After running the update operation in the previous example against an empty DatasetGraphNoSql, running the SPARQL query `SELECT ?s ?p ?o WHERE {?s ?p ?o}` generates the following response:

```
-----------------------------------------------------
| s                    | p                    | o  |
=====================================================
| <http://example/anne> | <http://example/age> | 30 |
-----------------------------------------------------
```

Using the same data, running the SPARQL query `SELECT ?s ?p ?o ?g where {GRAPH ?g {?s ?p ?o}}` generates the following response:

```
-------------------------------------------------------------------
| s                     | p                          | o    |
===================================================================
| <http://example/peter> | <http://example/birthyear> | 1982 |
-------------------------------------------------------------------
--------------------------------
|             g                |
================================
| <http://example/graph> |
--------------------------------
```

In addition to using the Java API for SPARQL Update operations, you can configure Apache Jena Joseki to accept SPARQL Update operations by removing the comment (##) characters at the start of the following lines in the joseki-config.ttl file.

```
## <#serviceUpdate>
## rdf:type joseki:Service ;
## rdfs:label "SPARQL/Update" ;
## joseki:serviceRef "update/service" ;
## # dataset part
## joseki:dataset <#oracle>;
## # Service part.
## # This processor will not allow either the protocol,
## # nor the query, to specify the dataset.
## joseki:processor joseki:ProcessorSPARQLUpdate
## .
##
## <#serviceRead>
## rdf:type joseki:Service ;
## rdfs:label "SPARQL" ;
## joseki:serviceRef "sparql/read" ;
## # dataset part
## joseki:dataset <#oracle> ; ## Same dataset
## # Service part.
## # This processor will not allow either the protocol,
## # nor the query, to specify the dataset.
## joseki:processor joseki:ProcessorSPARQL_FixedDS ;
## .
```

After you edit the `joseki-config.ttl` file, you must restart the Apache Jena Joseki Web application. You can then try a simple update operation, as follows:

1. In your browser, go to: http://<hostname>:7001/joseki/update.html
2. Type or paste the following into the text box:

```
PREFIX example: <http://example/>
INSERT DATA {
GRAPH <http://example/g1> { example:peter example:birthyear 1970  }
}
```

3. Click `Perform SPARQL Update`.

To verify that the update operation was successful, go to http://<hostname>:7001/joseki and enter the following query:

```
SELECT *
WHERE
{ GRAPH <http://example/g1>}{?s ?p ?o}}
```

The response should contain the following triple:

```
<http://example/peter> <http://example/birthyear> "1970"
```

A SPARQL Update can also be sent using an HTTP POST operation to the http://<hostname>:7001/joseki/update/service. For example, you can use cURL (http://en.wikipedia.org/wiki/CURL) to send an HTTP POST request to perform the update operation:

```
curl --data "request=PREFIX%20%3A%20%3Chttp%3A%2F%2Fexample \
        %2F%3E%20%0AINSERT%20DATA%20%7B%0A%20%20GRAPH%20%3 \
        Chttp%3A%2F%2Fexample%2Fg1%3E%20%7B%20%3Ar%20%3Ap%20 \
        888%20%7D%0A%7D%0A" \
        http://hostname:7001/joseki/update/service
```

In the preceding example, the URL encoded string is a simple INSERT operation into a named graph. After decoding, it reads as follows:

```
PREFIX : <http://example/>
INSERT DATA {
GRAPH <http://example/g1> { :r :p 888 }
```

Chapter 7. Inference on an RDF Graph

The RDF Graph feature supports RDF Schema (RDFS) and Web Ontology Language (OWL) inference through Apache Jena OntModel APIs. It also has the ability to support other memory-based third party reasoners, such as Pellet and TrOWL.

Use Jena OntModel APIs

Apache Jena provides a set of Java APIs including `Reasoner`, `ReasonerFactory`, `InfModel`, `OntModelSpec`, OntModel and more. Refer to http://jena.apache.org/documentation/inference/index.html for details. The following example describes how to use `OWL_MEM_RULE_INF` to build an OntologyModel on top of an OracleModelNoSql instance. The inference results are added to an in-memory Jena Model.

```
import java.io.PrintStream;
import java.util.Iterator;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.graph.*;
import oracle.rdf.kv.client.jena.*;

public class ExampleOntModel
{

  public static void main(String[] szArgs) throws Exception
  {

PrintStream psOut = System.out;

psOut.println("start");
String szStoreName = szArgs[0];
String szHostName  = szArgs[1];
String szHostPort  = szArgs[2];

// Create a connection to the Oracle NoSQL Database
OracleNoSqlConnection conn
                  = OracleNoSqlConnection.createInstance(szStoreName,
                                                         szHostName,
                                                         szHostPort);

// Create an OracleGraphNoSql object to handle the default graph
// and use it to create a Jena Model object.
Node graphNode = Node.createURI("http://example.org/graph1");
OracleGraphNoSql graph = new OracleNamedGraphNoSql(graphNode, conn);
Model model =
      OracleModelNoSql.createOracleModelNoSql(graphNode, conn);
```

```
// Clear model
model.removeAll();

Node sub = Node.createURI("http://sub/a");
Node pred = Node.createURI("http://pred/a");
Node obj = Node.createURI("http://obj/a");

// Add few axioms

Triple triple = Triple.create(sub, pred, obj);
graph.add(triple);

graph.add(Triple.create(pred,
    Node.createURI("http://www.w3.org/2000/01/rdf-schema#domain"),
    Node.createURI("http://C")));

graph.add(Triple.create(pred,
   Node.createURI("http://www.w3.org/1999/02/22-rdf-syntax-ns#type"),
   Node.createURI("http://www.w3.org/2002/07/owl#ObjectProperty")));

{
      // read it out
      Iterator it = GraphUtil.findAll(graph);

      while (it.hasNext()) {
        psOut.println("triple " + it.next().toString());
      }
    }

// Create an OntModel instance
OntModel om =
      ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF,
                                       model);

Model modelInMem = ModelFactory.createDefaultModel();
modelInMem.add(om);

    {
      Iterator it = GraphUtil.findAll(modelInMem.getGraph());
      while (it.hasNext()) {
                psOut.println("triple from OntModel " +
                it.next().toString());
      }
    }

model.close();
conn.close();
  }
}
```

For the above example, one can find the following triples from the output. The inference produces a correct classification of individual http://sub/a.

```
triple from OntModel http://sub/a @owl:sameAs http://sub/a
triple from OntModel http://sub/a @rdf:type rdfs:Resource
triple from OntModel http://sub/a @rdf:type owl:Thing
triple from OntModel http://sub/a @rdf:type http://C
triple from OntModel http://sub/a @http://pred/a http://obj/a
```

One can of course create an InferredNamedGraphNoSql object and add the contents from the OntModel into it. Furher details on storing inference triples using InferredNamedGraphNoSql class can be found in Example1c.java: Create an inferred graph and add/delete triples (page 39) and Example1d.java: Create an inferred graph and add/delete triples (page 41).

Use SPARQL Construct

Take RDFS entailment rule rdfs2 for example (see http://www.w3.org/TR/rdf-mt/#RDFSRules for details). One can use the following code snippet to materialize the hidden relationships.

```
String szConstruct =
      " PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
      + " PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "
      + " CONSTRUCT   { ?x rdf:type ?c } "
      + "     WHERE   { ?x ?p ?y . ?p rdfs:domain ?c } " ;
```

After adding the above code snippet into the ExampleOntModel source code (after the triples insertion), the following can be found from the output.

```
Rule execution results <ModelCom {http://sub/a @rdf:type http://C} | >
```

Use External Reasoner together with Jena APIs

There are a few external OWL reasoners that work with Apache Jena APIs. The following web pagedescribes a way to use Pellet together with Apache Jena APIs: http://clarkparsia.com/pellet/faq/using-pellet-in-jena/

TrOWL can also be used as a Apache Jena reasoner. Refer to the following page for details: http://trowl.eu/

Chapter 8. Quick Start for the RDF Graph Feature

This section provides examples for the major capabilities of the RDF Graph feature. Each example is self-contained: it typically creates a graph, adds triples, performs a query that may involve inference, displays the result, and drops the model.

These examples can be found in the `examples/` directory.

This section includes examples that do the following:

- Create a graph (or named graph) and insert/delete triples.
- Create an inferred graph (or inferred named graph) and insert/delete inferred triples belonging to a graph.
- Load a RDF file into the Oracle NoSQL Database.
- Run several SPARQL queries using a "family" ontology, including:
- SPARQL query features such as LIMIT, OFFSET, ASK, DESCRIBE, CONSTRUCT.
- RDF Graph feature query options like TIMEOUT, DOP, ORDERED, INF_ONLY, ASSERTED_ONLY.
- RDF Graph feature hints such as PLAN.
- Use the Apache Jena ARQ built-in function.
- Use a SELECT cast query.
- SPARQL Update requests to insert data into the Oracle NoSQL Database.
- Create a connection to an Oracle NoSQL Database using `OracleNoSqlConnection`.
- Use Oracle NoSQL Database connection pooling.
- Generate sampling data for a specified graph or data set.

To run a query, you must do the following:

1. Include the code in a Java source file. The examples used in this section are provided as files in the examples directory of the package.
2. Compile the Java source file. For example:
```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example.java
```
3. Run the compiled file. For example:
```
java -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example <store_name \
```

```
> <host_name> <host_port>
```

Example1.java: Create a default graph and add/delete triples

This example shows how to add/remove a set of triples over a default graph stored in an Oracle NoSQL Database.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example1 {

  public static void main(String[] args) throws Exception
  {

String szStoreName = args[0];
String szHostName = args[1];
String szHostPort = args[2];

OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// This object will handle operations over the default graph
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
graph.clearRepository(); //Clear the graph including inferred triples

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:Mary"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

String queryString = " select ?x ?y WHERE {?x <u:parentOf> ?y}";
System.out.println("Execute query " + queryString);

Model model = new OracleModelNoSql(graph);
Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }
```

```
finally {
      qexec.close();
    }

graph.delete(Triple.create(Node.createURI("u:John"),
                               Node.createURI("u:parentOf"),
                               Node.createURI("u:Mary")));

queryString = "select ?x ?y ?z WHERE {?x ?y ?z}";
System.out.println("Execute query " + queryString);

query = QueryFactory.create(queryString) ;
qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();

  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar \
Example1 <store_name> <host_name> <host_port>


Execute query select ?x ?y WHERE {?x <u:parentOf> ?y}
-----------------------
| x        | y        |
=======================
| <u:Mary> | <u:Jack> |
| <u:John> | <u:Mary> |
```

```
----------------------


Execute query select ?x ?y ?z WHERE {?x ?y ?z}
-------------------------------------
| x        | y            | z        |
=====================================
| <u:Mary> | <u:parentOf> | <u:Jack> |
-------------------------------------
```

Example1b.java: Create a named graph and add/delete triples

This example describes how to add/remove a set of triples over a named graph stored in an Oracle NoSQL Database.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example1b {

  public static void main(String[] args) throws Exception
  {

String szStoreName = args[0];
String szHostName = args[1];
String szHostPort = args[2];
String szGraphName = args[3];

OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// This object will handle operations over the named graph
OracleGraphNoSql graph
                 = new OracleNamedGraphNoSql(szGraphName, conn);
// Clear the named graph including inferred triples
graph.clearRepository();

// Add triples
graph.add(Triple.create(Node.createURI("u:John"),
                            Node.createURI("u:parentOf"),
                            Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:Mary"),
                            Node.createURI("u:parentOf"),
                            Node.createURI("u:Jack")));
```

```
String queryString = " select ?x ?y WHERE {?x <u:parentOf> ?y}";
System.out.println("Execute query " + queryString);

Model model = new OracleModelNoSql(graph);
Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

graph.delete(Triple.create(Node.createURI("u:John"),
                               Node.createURI("u:parentOf"),
                               Node.createURI("u:Mary")));

queryString = "select ?x ?y ?z WHERE {?x ?y ?z}";
System.out.println("Execute query " + queryString);

query = QueryFactory.create(queryString);
qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar: ./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1b.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
```

```
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1b <store_name> \
<host_name> <host_port> <graph_name>


Execute query select ?x ?y WHERE {?x <u:parentOf>?y}
----------------------
| x        | y        |
======================
| <u:Mary> | <u:Jack> |
| <u:John> | <u:Mary> |
----------------------


Execute query select ?x ?y ?z WHERE {?x ?y ?z}
-------------------------------------
| x        | y            | z        |
=====================================
| <u:Mary> | <u:parentOf> | <u:Jack> |
-------------------------------------
```

Example1c.java: Create an inferred graph and add/delete triples

This example describes how to add inferred triples for a default graph. Inferred triples are managed in the RDF Graph feature through an InferredGraphNoSql object. Triples in the inferred graph are tagged with an integer rulebase ID.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example1c {

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

// the rulebase id used for inferred triples
int iRuleBaseId     = Integer.parseInt(args[3]);

OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);
```

```
// This object handle a model associated to a default graph
Model model = OracleModelNoSql.createOracleDefaultModelNoSql(conn);
OracleGraphNoSql graph = (OracleGraphNoSql) model.getGraph();

model.removeAll(); // removes all the triples from the associated
                   // model, this will remove all asserted and
                   // inferred triples

graph.add(Triple.create(Node.createURI("u:John"),
                           Node.createURI("u:parentOf"),
                           Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                           Node.createURI("u:parentOf"),
                           Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Amy"),
                           Node.createURI("u:parentOf"),
                           Node.createURI("u:Jack")));

// This object handles all the inferred triples of the default graph
// produced with rulebase ID
InferredGraphNoSql inferredGraph =
                          new InferredGraphNoSql(conn, iRuleBaseId);

inferredGraph.add(Triple.create(Node.createURI("u:Jack"),
                               Node.createURI("u:siblingOf"),
                               Node.createURI("u:Mary")));

inferredGraph.close();


String prefix =
         " PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#" +
         "include_rulebase_id=" + iRuleBaseId + ">";

String szQuery = prefix + " select ?x ?y ?z WHERE {?x ?y ?z} ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }
finally {
```

```
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1c.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1c <store_name> \
<host_name> <host_port> <rule_base_id>


Execute query select ?x ?y ?z WHERE {?x ?y ?z}
-------------------------------------
| x        | y            | z        |
=====================================
| <u:Mary> | <u:siblingOf> | <u:Mary> |
| <u:John> | <u:parentOf>  | <u:Jack> |
| <u:John> | <u:parentOf>  | <u:Mary> |
| <u:Amy>  | <u:parentOf>  | <u:Jack> |
-------------------------------------
```

Example1d.java: Create an inferred graph and add/delete triples

This example describes how to add inferred triples for a named graph. Inferred triples are managed in the RDF Graph feature through an InferredNamedGraphNoSql object. Triples in the inferred graph are tagged with an integer rulebase ID.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example1d {

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
```

```
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];

// the rulebase id used for inferred triples
int iRuleBaseId     = Integer.parseInt(args[4]);

OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// This object handle a model associated to a named graph
Model model =
          OracleModelNoSql.createOracleModelNoSql(szModelName, conn);
OracleGraphNoSql graph = (OracleGraphNoSql) model.getGraph();

model.removeAll(); // removes all the triples from the associated
                   // model this will remove all asserted and
                   // inferred triples

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Amy"),
          Node.createURI("u:parentOf"),
          Node.createURI("u:Jack")));

// This object handles all the inferred triples of
// the named graph produced with rulebase ID
InferredGraphNoSql inferredGraph =
                  new InferredNamedGraphNoSql(szModelName,
                                              conn,
                                              iRuleBaseId);

inferredGraph.add(Triple.create(Node.createURI("u:Jack"),
                                Node.createURI("u:siblingOf"),
                                Node.createURI("u:Mary")));

inferredGraph.close();

String prefix =
          "PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#" +
          "include_rulebase_id=" + iRuleBaseId + ">";
```

```
String szQuery = prefix + " select ?x ?y ?z WHERE {?x ?y ?z} ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1d.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example1d <store_name> \
<host_name> <host_port> \<graph_name> < \
rule_base_id>


Execute query select ?x ?y ?z WHERE {?x ?y ?z}
--------------------------------------
| x        | y            | z        |
======================================
| <u:Jack> | <u:siblingOf> | <u:Mary> |
| <u:John> | <u:parentOf>  | <u:Jack> |
| <u:John> | <u:parentOf>  | <u:Mary> |
| <u:Amy>  | <u:parentOf>  | <u:Jack> |
--------------------------------------
```

Example2.java: Load an RDF file

This example loads an RDF file into Oracle NoSQL Database. The example also queries for all the quads stored in the Oracle NoSQL Database.

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example2
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

System.out.println("Create Oracle NoSQL connection");
OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

System.out.println("Create Oracle NoSQL datasetgraph");
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear datasetgraph
datasetGraph.clearRepository();

// Load data from file into the Oracle NoSQL Database
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                           "http://example.org/"); // base URI

// Create dataset from Oracle NoSQL datasetgraph to execute
Dataset ds = DatasetImpl.wrap(datasetGraph);

String szQuery = "select * where { graph ?g { ?s ?p ?o }  }";
System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
```

```
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar: ./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example2.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar: ./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example2 <store_name> \
<host_name> <host_port>

Execute query select * where { graph ?g { ?s ?p ?o } }
-----------------------------------------
| s                                     |
=========================================
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| _:b0                                  |
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| <http://example.org/bob/foaf.rdf#me>   |
| <http://example.org/bob/foaf.rdf#me>   |
| <http://example.org/bob/foaf.rdf#me>   |
-----------------------------------------
-----------------------------------------------------
| p                                                 |
=====================================================
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://xmlns.com/foaf/0.1/homepage>              |
| <http://xmlns.com/foaf/0.1/knows>                 |
| <http://www.w3.org/2000/01/rdf-schema#seeAlso>    |
```

```
| <http://xmlns.com/foaf/0.1/name>                   |
| <http://xmlns.com/foaf/0.1/name>                   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://xmlns.com/foaf/0.1/homepage>               |
| <http://xmlns.com/foaf/0.1/name>                   |
-----------------------------------------------------
----------------------------------------
| o                                    |
========================================
| <http://xmlns.com/foaf/0.1/Person>   |
| <http://xmlns.com/foaf/0.1/Person>   |
| <http://example.org/bob/>            |
| _:b0                                 |
| <http://example.org/bob/foaf.rdf>    |
| "Bob"                                |
| "Alice"                              |
| <http://xmlns.com/foaf/0.1/Person>   |
| <http://example.org/bob/>            |
| "Bob"                                |
----------------------------------------
----------------------------------------
| g                                    |
========================================
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/alice/foaf.rdf>  |
| <http://example.org/bob/foaf.rdf>    |
| <http://example.org/bob/foaf.rdf>    |
| <http://example.org/bob/foaf.rdf>    |
----------------------------------------
```

Concurrent RDF data loading

This example loads an RDF file into Oracle NoSQL Database using parallel loading (multiple threads). To use parallel loading, you can specify the degree of parallelism (number of threads used to load data as well as the batch size of the bucket of triples managed by each thread. The example also queries for all the quads stored in the Oracle NoSQL Database.

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example2b
{
```

```
  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
int iBatchSize      = Integer.parseInt(args[3]);
int iDOP            = Integer.parseInt(args[4]);

System.out.println("Create Oracle NoSQL connection");
OracleNoSqlConnection conn
= OracleNoSqlConnection.createInstance(szStoreName,
                                       szHostName,
                                       szHostPort);

System.out.println("Create Oracle NoSQL datasetgraph");
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear datasetgraph
datasetGraph.clearRepository();

// Load data from file into the Oracle NoSQL Database
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                       "http://example.org",
                       iBatchSize, // batch size
                       iDOP); // degree of parallelism

// Create dataset from Oracle NoSQL datasetgraph to execute
Dataset ds = DatasetImpl.wrap(datasetGraph);

String szQuery = "select * where { graph ?g { ?s ?p ?o }  }";
System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }
```

```
ds.close();
conn.dispose();
 }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example2b.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example2b <store_name> \
<host_name> <host_port> <batch_size> <dop>

Execute query select * where { graph ?g { ?s ?p ?o } }

-----------------------------------------
| s                                     |
=========================================
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| _:b0                                  |
| _:b0                                  |
| <http://example.org/alice/foaf.rdf#me> |
| <http://example.org/bob/foaf.rdf#me>   |
| <http://example.org/bob/foaf.rdf#me>   |
| <http://example.org/bob/foaf.rdf#me>   |
-----------------------------------------
-----------------------------------------------------
| p                                                 |
=====================================================
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://xmlns.com/foaf/0.1/homepage>              |
| <http://xmlns.com/foaf/0.1/knows>                 |
| <http://www.w3.org/2000/01/rdf-schema#seeAlso>    |
| <http://xmlns.com/foaf/0.1/name>                  |
| <http://xmlns.com/foaf/0.1/name>                  |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://xmlns.com/foaf/0.1/homepage>              |
| <http://xmlns.com/foaf/0.1/name>                  |
-----------------------------------------------------
-----------------------------------------
```

```
| o                                        |
==========================================
| <http://xmlns.com/foaf/0.1/Person>     |
| <http://xmlns.com/foaf/0.1/Person>     |
| <http://example.org/bob/>              |
| _:b0                                   |
| <http://example.org/bob/foaf.rdf>      |
| "Bob"                                  |
| "Alice"                                |
| <http://xmlns.com/foaf/0.1/Person>     |
| <http://example.org/bob/>              |
| "Bob"                                  |
------------------------------------------
------------------------------------------
| g                                        |
==========================================
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/alice/foaf.rdf>    |
| <http://example.org/bob/foaf.rdf>      |
| <http://example.org/bob/foaf.rdf>      |
| <http://example.org/bob/foaf.rdf>      |
------------------------------------------
```

Example4.java: Query family relationships on a named graph

This example specifies that John loves Mary (included in the default graph), and it selects and displays the subject and object in each fatherOf relationship (as JSON output). Example4b.java in the RDF Graph feature describes the same exercise using a named graph.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example4
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

System.out.println("Create Oracle NoSQL connection");
```

```
OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

System.out.println("Create Oracle NoSQL model");
Model model = OracleModelNoSql.createOracleDefaultModelNoSql(conn);

System.out.println("Clear model");
model.removeAll();


System.out.println("Add triples");
model.getGraph().add(
           Triple.create(Node.createURI("http://example.com/John"),
                         Node.createURI("http://example.com/loves"),
                         Node.createURI("http://example.com/Mary")));


String queryString  =
              " select ?person1 ?person2 "                    +
              " where "                                       +
              " { ?person1 <http://example.com/loves> ?person2 }";

System.out.println("Execute query " + queryString);

Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.outputAsJSON(System.out, results);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example4.java
```

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example4b <store_name> \
<host_name> <host_port>

Execute query select ?person1 ?person2 where { ?person1
<http://example.com/loves> ?person2 }

{
  "head": {
    "vars": [ "person1" , "person2" ]
  } ,
  "results": {
    "bindings": [
      {
        "person1": { "type": "uri" , "value": "http://example.com/John" } ,
        "person2": { "type": "uri" , "value": "http://example.com/Mary" }
      }
    ]
  }
}
```

Example5.java: SPARQL query with JOIN_METHOD

This example shows a SPARQL query with additional features including the selection of a join_method (JOIN_METHOD={nl, hash}) used to select the join method to use in operations over query patterns. It loads the quads contained in RDF file example.nt located in the examples directory of the RDF Graph feature that assert the following:

- In graph <http://example.org/alice/foaf.rdf>:
  - Alice is a Person.
  - Alice's name is "Alice".
  - Alice knows Bob.
  - Bob has a home page with URL http://example.org/bob.
  - To see more details on Bob, refer to http://example.org/alice/foaf.rdf.
- In graph <http://example.org/bob/foaf.rdf>:
  - Bob is a Person.
  - Bob's name is "Bob".
  - Bob has a home page with URL http://example.org/bob.

It then finds all the names of the people in graph <http://example.org/alice/foaf.rdf> who knows another person, using nested loop join operations to merge bindings retrieved between the query patterns in the SPARQL query.

```
import com.hp.hpl.jena.query.*;
import org.openjena.riot.Lang;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import oracle.rdf.kv.client.jena.*;

public class Example5
{

public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

// Create connection
OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// Create the datasetgraph
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear dataset
datasetGraph.clearRepository();

Dataset ds = DatasetImpl.wrap(datasetGraph);

// Load data from file into the dataset
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                           "http://example.com"); //base URI

// change hint to hash to test hash join, or remove to use default
// join settings
String szQuery =
" PREFIX ORACLE_SEM_FS_NS: "                          +
" <http://oracle.com/semtech#join_method=nl>"         +
" PREFIX foaf: <http://xmlns.com/foaf/0.1/>"          +
" SELECT ?name1 ?name2 "                              +
" WHERE { "                                           +
"   graph <http://example.org/alice/foaf.rdf> { "     +
```

```
"     ?person1 foaf:knows ?person2 . "               +
"     ?person1 foaf:name ?name1 . "                  +
"     ?person2 foaf:name ?name2 . "                  +
"   } "                                              +
" } ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5 <store_name> \
<host_name> <host_port>
------------------
| name1   | name2 |
==================
| "Alice" | "Bob" |
------------------
```

You can test hash join selection by modifying the following line in the code. The output of this Java class will be the same as the one presented before.

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#join_method=hash>
```

Example5b.java: SPARQL query with ORDERED query option

This example shows the SPARQL query from Example5.java: SPARQL query with JOIN_METHOD (page 51) with additional features including the ORDERED query option, where you can specify that the order in which query patterns are executed, is based on the order in which they are defined in the SPARQL query.

```
import com.hp.hpl.jena.query.*;
import org.openjena.riot.Lang;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import oracle.rdf.kv.client.jena.*;

public class Example5b
{

  public static void main(String[] args) throws Exception
  {

    String szStoreName = args[0];
    String szHostName = args[1];
    String szHostPort = args[2];

    System.out.println("create connection");
    OracleNoSqlConnection conn
        = OracleNoSqlConnection.createInstance(szStoreName,
                                               szHostName,
                                               szHostPort);

    System.out.println("Create datasetgraph");
    OracleGraphNoSql graph = new OracleGraphNoSql(conn);
    DatasetGraphNoSql datasetGraph
                        = DatasetGraphNoSql.createFrom(graph);

    // Close graph, as it is no longer needed
    graph.close();

    System.out.println("Clear dataset");
    datasetGraph.clearRepository();


    System.out.println("Load data from file into DatasetGraphNoSql");
    DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                           "http://example.com");

    Dataset ds = DatasetImpl.wrap(datasetGraph);

    String queryString =
        " PREFIX ORACLE_SEM_FS_NS: " +
        " <http://oracle.com/semtech#ordered>" +
        " PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
```

```
        " SELECT ?name1 ?name2 " +
        " WHERE { " +
        " graph <http://example.org/alice/foaf.rdf> { " +
        "   ?person1 foaf:name ?name1 . " +
        "   ?person1 foaf:knows ?person2 . " +
        "   ?person2 foaf:name ?name2 . " +
        " } } ";

    System.out.println("Execute query " + queryString);

    Query query = QueryFactory.create(queryString);
    QueryExecution qexec = QueryExecutionFactory.create(query, ds);

    try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

    finally {
      qexec.close();
    }

    ds.close();
    conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar: ./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar:. \
/jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5b.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5b <store_name> \
<host_name> <host_port>

-------------------
| name1   | name2 |
===================
| "Alice" | "Bob" |
-------------------
```

Example5c.java: SPARQL query with TIMEOUT and GRACEFUL TIMEOUT

This example shows the SPARQL query from Example5.java: SPARQL query with JOIN_METHOD (page 51) with additional features including a timeout setting (TIMEOUT=1, in seconds). You can modify this code by adding a graceful timeout setting (BEST_EFFORT_QUERY=T) in order to avoid getting an error and retrieve all triples found until timeout.

```
import com.hp.hpl.jena.query.*;
import org.openjena.riot.Lang;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import oracle.rdf.kv.client.jena.*;

public class Example5c
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

// create connection
OracleNoSqlConnection conn
= OracleNoSqlConnection.createInstance(szStoreName,
                                       szHostName,
                                       szHostPort);

// Create datasetgraph
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear the dataset
datasetGraph.clearRepository();

Dataset ds = DatasetImpl.wrap(datasetGraph);

// Load data from file into the dataset
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                       "http://example.com");

// Add a hint best_effort_query=t to use a graceful timeout policy
String szQuery =
" PREFIX ORACLE_SEM_FS_NS: "                          +
```

```
" <http://oracle.com/semtech#timeout=1>"            +
" PREFIX foaf: <http://xmlns.com/foaf/0.1/>"        +
" SELECT ?name1 ?name2 ?homepage2 "                 +
" WHERE { "                                         +
"   graph <http://example.org/alice/foaf.rdf> { "   +
"     ?person1 foaf:knows ?person2 . "              +
"     ?person1 foaf:name ?name1 . "                 +
"     ?person2 foaf:name ?name2 . "                 +
"     ?person2 foaf:homepage ?homepage2 . "         +
"   } "                                             +
" } ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command if no graceful timeout is set.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar: ./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5c.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar:
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5c <store_name> \
<host_name> <host_port>
Exception in thread "main" com.hp.hpl.jena.shared.JenaException:
com.hp.hpl.jena.shared.JenaException: Timeout exceeded, user requested to
end data retrieval
```

The following represents the expected output of the java command if a graceful timeout is set.

```
-----------------------------------------------
| name1   | name2 | homepage2                 |
===============================================
| "Alice" | "Bob" | <http://example.org/bob/> |
-----------------------------------------------
```

Example5d.java: SPARQL query with DOP

This example shows the SPARQL query from Example5.java: SPARQL query with JOIN_METHOD (page 51) with additional features including a parallel execution setting (DOP=4).

```
import com.hp.hpl.jena.query.*;
import org.openjena.riot.Lang;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import oracle.rdf.kv.client.jena.*;

public class Example5d
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

System.out.println("create connection");
OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// Create datasetgraph
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear dataset
datasetGraph.clearRepository();

Dataset ds = DatasetImpl.wrap(datasetGraph);

// Load data from file into the dataset
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                       "http://example.com");

String szQuery =
```

```
" PREFIX ORACLE_SEM_FS_NS: "                          +
" <http://oracle.com/semtech#dop=4>"                  +
" PREFIX foaf: <http://xmlns.com/foaf/0.1/>"          +
" SELECT ?name1 ?name2 ?homepage2 "                   +
" WHERE { "                                           +
"   graph <http://example.org/alice/foaf.rdf> { "     +
"     ?person1 foaf:knows ?person2 . "                +
"     ?person1 foaf:name ?name1 . "                   +
"     ?person2 foaf:name ?name2 . "                   +
"     ?person2 foaf:homepage ?homepage2 . "           +
"   } "                                               +
" } ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
}

finally {
      qexec.close();
}

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5d.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5d <store_name> \
 <host_name> <host_port>
-----------------------------------------------
| name1   | name2 | homepage2                 |
===============================================
| "Alice" | "Bob" | <http://example.org/bob/> |
-----------------------------------------------
```

Example5e.java: SPARQL query with INFERENCE/ASSERTED ONLY hints

Example 8.12 shows the SPARQL query with additional features including an inference only setting (INF_ONLY). It inserts triples that assert the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Mary.
- Jack is a sibing of Mary (specified as inferred).

It then finds all the triples in the Oracle NoSQL Database. Example 5f in the RDF Graph feature package describes the same exercise using an asserted only setting (ASSERTED_ONLY). Note that Example 5f is not shown in this manual.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example5e
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
int iRuleBaseId     = Integer.parseInt(args[3]);

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
     = OracleNoSqlConnection.createInstance(szStoreName,
                                            szHostName,
                                            szHostPort);

// Create model from default graph
Model model = OracleModelNoSql.createOracleDefaultModelNoSql(conn);
OracleGraphNoSql graph = (OracleGraphNoSql) model.getGraph();

// Clear model
model.removeAll();

// Add triples

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
```

```
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Amy"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

// Create Oracle NoSQL inferred graph

InferredGraphNoSql inferredGraph =
                        new InferredGraphNoSql(conn,
                                               iRuleBaseId);

// Add inferred triples
inferredGraph.add(Triple.create(Node.createURI("u:Jack"),
                                    Node.createURI("u:siblingOf"),
                                    Node.createURI("u:Mary")));

// Close inferred graph;
inferredGraph.close();

String prefix = " PREFIX ORACLE_SEM_FS_NS: " +
                " <http://oracle.com/semtech#" +
                "include_rulebase_id=" + iRuleBaseId +
                ",inf_only>";
String szQuery = prefix + " select ?x ?y ?z WHERE {?x ?y ?z} ";
System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }


model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command with an inference only setting.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-
2.9.4.jar:./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5e.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5e <store_name> \
<host_name> <host_port> <rule_baseID>

Execute query PREFIX ORACLE_SEM_FS_NS:
<http://oracle.com/semtech#inf_only>
select ?x ?y ?z WHERE {?x ?y ?z}

---------------------------------------
| x        | y             | z        |
=======================================
| <u:Jack> | <u:siblingOf> | <u:Mary> |
---------------------------------------
```

The following represents the expected output of the java command if an asserted only setting is specified.

```
-------------------------------------
| x        | y            | z        |
=====================================
| <u:John> | <u:parentOf> | <u:Jack> |
| <u:John> | <u:parentOf> | <u:Mary> |
| <u:Amy>  | <u:parentOf> | <u:Jack> |
-------------------------------------
```

Example5g.java: SPARQL query with PLAN query hint

This example shows the SPARQL query from Example5.java: SPARQL query with JOIN_METHOD (page 51) with additional features including a PLAN setting (PLAN=encoded_plan), where you can specify the execution plan associated to the query patterns of this query. Further details on query hints can be found in Query hints (page 25).

```
import com.hp.hpl.jena.query.*;
import org.openjena.riot.Lang;
import com.hp.hpl.jena.sparql.core.DatasetImpl;
import java.net.URLEncoder;
import oracle.rdf.kv.client.jena.*;

public class Example5g
{

  public static void main(String[] args) throws Exception
```

```
  {

String szStoreName = args[0];
String szHostName = args[1];
String szHostPort = args[2];

    // Create connection
    OracleNoSqlConnection conn
        = OracleNoSqlConnection.createInstance(szStoreName,
                                               szHostName,
                                               szHostPort);

// Create datasetgraph
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear dataset
datasetGraph.clearRepository();

// Load data from file into the dataset
DatasetGraphNoSql.load("example.nt", Lang.NQUADS, conn,
                       "http://example.com");


Dataset ds = DatasetImpl.wrap(datasetGraph);

String plan = URLEncoder.encode("((qp2 qp3 NLJ) qp1 NLJ)", "UTF-8");

String queryString =
" PREFIX ORACLE_SEM_HT_NS: "                          +
" <http://oracle.com/semtech#plan=" + plan + ">"      +
" PREFIX foaf: <http://xmlns.com/foaf/0.1/>"          +
" SELECT ?name1 ?name2 "                              +
" WHERE { " +
" graph <http://example.org/alice/foaf.rdf> { "       +
"   ?person1 foaf:knows ?person2 . "                  +
"   ?person1 foaf:name ?name1 . "                     +
"   ?person2 foaf:name ?name2 . "                     +
" } } ";

System.out.println("Execute query " + queryString);

Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
```

```
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5g.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example5g <store_name> \
<host_name> <host_port>
-------------------
| name1   | name2 |
===================
| "Alice" | "Bob" |
-------------------
```

Example6.java: SPARQL ASK query

Thsi example shows a SPARQL ASK query. It inserts a triple that postulates that John is a parent of Mary. It then finds whether John is a parent of Mary.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import oracle.rdf.kv.client.jena.*;

public class Example6
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];
```

```
// Create Oracle NoSQL connection
OracleNoSqlConnection conn
      = OracleNoSqlConnection.createInstance(szStoreName,
                                             szHostName,
                                             szHostPort);


// Create model from named graph
OracleModelNoSql model =
OracleModelNoSql.createOracleModelNoSql(szModelName,
                                        conn);
// Clear model
model.removeAll();

// Get graph from model
OracleGraphNoSql graph = model.getGraph();

// Add triples

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

String szQuery =
          " ASK { <u:John> <u:parentOf> <u:Mary> } ";

System.out.println("Execute ASK query " + szQuery);

Query query = QueryFactory.create(szQuery) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

boolean b = qexec.execAsk();
System.out.println("Ask result = " + ((b)?"TRUE":"FALSE"));

// Close objects
qexec.close();
model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example6.java
```

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example6 <store_name> \
<host_name> <host_port> <graph_name>

Execute ASK query ASK { <u:John> <u:parentOf> <u:Mary> }
Ask result = TRUE
```

Example7.java: SPARQL Describe query

This example shows a SPARQL DESCRIBE query. It inserts triples that assert the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Jack.

It then finds all relationships that involve any parents of Jack.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example7
{

  public static void main(String[] args) throws Exception
  {

String szStoreName   = args[0];
String szHostName    = args[1];
String szHostPort    = args[2];
String szModelName   = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
          = OracleNoSqlConnection.createInstance(szStoreName,
                                                 szHostName,
                                                 szHostPort);

// Create model from named graph
OracleModelNoSql model =
               OracleModelNoSql.createOracleModelNoSql(szModelName,
                                                       conn);

// Clear model
model.removeAll();
```

```
// Get graph from model
OracleGraphNoSql graph = model.getGraph();

// Add triples

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Amy"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

String szQuery =
            "DESCRIBE ?x WHERE {?x <u:parentOf> <u:Jack>}";

System.out.println("Execute describe query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

Model describeModel = qexec.execDescribe();

System.out.println("Describe result = " + describeModel.toString());

qexec.close();
describeModel.close();

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example7.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example7 <store_name> \
<host_name> <host_port> <graph_name>
```

```
Execute describe query DESCRIBE ?x WHERE {?x <u:parentOf> <u:Jack>}
Describe result = <ModelCom   {u:Amy @u:parentOf u:Jack; u:John
@u:parentOf u:Mary; u:John @u:parentOf u:Jack} | >
```

Example8.java: SPARQL Construct query

This example shows a SPARQL CONSTRUCT query. It inserts triples that assert the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Jack.
- Each parent loves all of his or her children.

It then constructs an RDF graph with information about who loves whom.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example8
{
  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];

System.out.println("Create Oracle NoSQL connection");
OracleNoSqlConnection conn
= OracleNoSqlConnection.createInstance(szStoreName,
                                       szHostName,
                                       szHostPort);

System.out.println("Create Oracle NoSQL model");

OracleModelNoSql model =
                OracleModelNoSql.createOracleModelNoSql(szModelName,
                                                        conn);

System.out.println("Clear model");
model.removeAll();

System.out.println("Get graph from model");
OracleGraphNoSql graph = model.getGraph();
```

```
System.out.println("Add triples");

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Amy"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

String szQuery = "CONSTRUCT { ?s <u:loves> ?o } " +
                 "WHERE {?s <u:parentOf> ?o }";

System.out.println("Execute construct query " + szQuery);

Query query = QueryFactory.create(szQuery) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);

Model constructModel = qexec.execConstruct();
System.out.println("Construct result = " + constructModel.toString());

qexec.close();
constructModel.close();

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example8.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example8 <store_name> \
<host_name> <host_port> <graph_name>
Execute construct query CONSTRUCT { ?s <u:loves> ?o }
WHERE {?s <u:parentOf> ?o }
Construct result = <ModelCom   {u:Amy @u:loves u:Jack;
u:John @u:loves u:Mary; u:John @u:loves u:Jack} | >
```

Example9.java: SPARQL OPTIONAL query

This example shows a SPARQL OPTIONAL query. It inserts triples that assert the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds parent-child relationships, optionally including any grandchild (gkid) relationships.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import oracle.rdf.kv.client.jena.*;

public class Example9
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
          = OracleNoSqlConnection.createInstance(szStoreName,
                                                 szHostName,
                                                 szHostPort);

// Create model for named graph
OracleModelNoSql model
              = OracleModelNoSql.createOracleModelNoSql(szModelName,
                                                        conn);

// Clear model
model.removeAll();

// Get graph from model
OracleGraphNoSql graph = model.getGraph();

// Add triples
graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
```

```
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Mary"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jill")));

String szQuery = " SELECT ?s ?o ?gkid "                     +
                 " WHERE { "                                +
                 "    ?s <u:parentOf> ?o  . "               +
                 "    OPTIONAL {?o <u:parentOf> ?gkid } "   +
                 " } ";


System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example9.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example9 <store_name> \
<host_name> <host_port> <graph_name>
-----------------------------------
| s        | o        | gkid     |
===================================
| <u:Mary> | <u:Jill> |          |
```

```
| <u:John> | <u:Jack> |          |
| <u:John> | <u:Mary> | <u:Jill> |
---------------------------------
```

Example10.java: SPARQL query with LIMIT and OFFSET

This example shows a SPARQL query with LIMIT and OFFSET. It inserts triples that assert the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds one parent-child relationship (LIMIT 1), skipping the first two parent-child relationships encountered (OFFSET 2), and optionally includes any grandchild (gkid) relationships for the one found.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example10
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
          = OracleNoSqlConnection.createInstance(szStoreName,
                                                 szHostName,
                                                 szHostPort);

// Create model from named graph
Model model =
      OracleModelNoSql.createOracleModelNoSql(szModelName, conn);
OracleGraphNoSql graph = (OracleGraphNoSql) model.getGraph();

// Clear graph
graph.clearRepository();


// Add triples
```

```
graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));

graph.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));

graph.add(Triple.create(Node.createURI("u:Mary"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jill")));


String szQuery = " SELECT ?s ?o ?gkid "                          +
                 " WHERE { ?s <u:parentOf> ?o . "                +
                 " OPTIONAL {?o <u:parentOf> ?gkid }} "          +
                 " LIMIT 1 OFFSET 2";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example10.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example10 <store_name> \
<host_name> <host_port> <graph_name>
```

```
Execute query  SELECT ?s ?o ?gkid  WHERE { ?s <u:parentOf> ?o .
OPTIONAL {?o <u:parentOf> ?gkid }}  LIMIT 1 OFFSET 2\
---------------------------------
| s        | o        | gkid     |
=================================
| <u:John> | <u:Mary> | <u:Jill> |
---------------------------------
```

Example11.java: SPARQL query with SELECT Cast

This example "converts" two Fahrenheit temperatures (18.1 and 32.0) to Celsius temperatures.

```
import com.hp.hpl.jena.update.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;

public class Example11
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
           = OracleNoSqlConnection.createInstance(szStoreName,
                                                  szHostName,
                                                  szHostPort);

// Create model from named graph
Model model =
      OracleModelNoSql.createOracleModelNoSql(szModelName, conn);

// Clear model
model.removeAll();


String insertString =
" PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> "          +
" INSERT DATA "                                              +
" { "                                                        +
"     <u:Object1> <u:temp> \"18.1\"^^xsd:float ; "           +
"     <u:name> \"Foo... \" . "                               +
"     <u:Object2> <u:temp> \"32.0\"^^xsd:float ; "           +
```

```
"     <u:name> \"Bar... \" . "                                    +
" } ";


System.out.println("Execute insert action " + insertString);
UpdateAction.parseExecute(insertString, model);

String szQuery =
" PREFIX fn: <http://www.w3.org/2005/xpath-functions#> "         +
" SELECT ?subject ?temp ((?temp - 32.0)*5/9 as ?celsius_temp) "   +
" WHERE { ?subject <u:temp> ?temp } ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery, Syntax.syntaxARQ);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example11.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example11 <store_name> \
<host_name> <host_port> <graph_name>

Execute insert action  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
INSERT DATA  {
<u:Object1> <u:temp> "18.1"^^xsd:float ; <u:name> "Foo... " .
<u:Object2> <u:temp> "32.0"^^xsd:float ; <u:name> "Bar... " . }
```

```
Execute query  PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
SELECT ?subject ?temp ((?temp - 32.0)*5/9 as ?celsius_temp)
WHERE { ?subject <u:temp> ?temp }
-----------------------------------------------------
| subject     | temp | celsius_temp                 |
=====================================================
| <u:Object1> | 18.1 | -7.72222222222222222222223 |
| <u:Object2> | 32   | 0.                           |
-----------------------------------------------------
```

Example12.java: SPARQL Involving Named Graphs

This example shows a query involving named graphs. It involves a default graph that has information about named graph URIs and their publishers. The query finds graph names, their publishers, and within each named graph finds the mailbox value using the foaf:mbox predicate.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.sparql.core.*;
import com.hp.hpl.jena.query.*;
import oracle.rdf.kv.client.jena.*;

public class Example12
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
       = OracleNoSqlConnection.createInstance(szStoreName,
                                              szHostName,
                                              szHostPort);

// Create Oracle NoSQL graph and dataset
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph, as it is no longer needed
graph.close();

// Clear dataset
datasetGraph.clearRepository();

// add data to the default graph
datasetGraph.add(new Quad(
```

```
        Quad.defaultGraphIRI, // specifies default graph
        Node.createURI("http://example.org/bob"),
        Node.createURI("http://purl.org/dc/elements/1.1/publisher"),
        Node.createLiteral("Bob Hacker")));

datasetGraph.add(new Quad(
        Quad.defaultGraphIRI, // specifies default graph
        Node.createURI("http://example.org/alice"),
        Node.createURI("http://purl.org/dc/elements/1.1/publisher"),
        Node.createLiteral("alice Hacker")));

// add data to the bob named graph
datasetGraph.add(new Quad(
        Node.createURI("http://example.org/bob"), // graph name
        Node.createURI("urn:bob"),
        Node.createURI("http://xmlns.com/foaf/0.1/name"),
        Node.createLiteral("Bob")));

datasetGraph.add(new Quad(
        Node.createURI("http://example.org/bob"), // graph name
        Node.createURI("urn:bob"),
        Node.createURI("http://xmlns.com/foaf/0.1/mbox"),
        Node.createURI("mailto:bob@example")));

// add data to the alice named graph
datasetGraph.add(new Quad(
        Node.createURI("http://example.org/alice"), // graph name
        Node.createURI("urn:alice"),
        Node.createURI("http://xmlns.com/foaf/0.1/name"),
        Node.createLiteral("Alice")));

datasetGraph.add(new Quad(
        Node.createURI("http://example.org/alice"), // graph name
        Node.createURI("urn:alice"),
        Node.createURI("http://xmlns.com/foaf/0.1/mbox"),
        Node.createURI("mailto:alice@example")));

Dataset ds = DatasetImpl.wrap(datasetGraph);

String szQuery = " PREFIX foaf: <http://xmlns.com/foaf/0.1/>"    +
" PREFIX dc: <http://purl.org/dc/elements/1.1/> "    +
" SELECT ?who ?graph ?mbox "                          +
" FROM NAMED <http://example.org/alice>"            +
" FROM NAMED <http://example.org/bob>"              +
" WHERE "                                            +
" { "                                                +
" ?graph dc:publisher ?who . "                       +
" GRAPH ?graph { ?x foaf:mbox ?mbox } "              +
" } ";
```

```
Query query = QueryFactory.create(szQuery);
QueryExecution qexec = QueryExecutionFactory.create(query, ds);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

ds.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example12.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example12 <store_name> \
<host_name> <host_port>

-------------------------------------------------------------------------
| who            | graph                       | mbox                  |
=========================================================================
| "Bob Hacker"   | <http://example.org/bob>    | <mailto:bob@example> |
|"alice Hacker"  | <http://example.org/alice> | <mailto:alice@example>|
-------------------------------------------------------------------------
```

Example13.java: SPARQL Query with ARQ Built-in Functions

This example inserts data about two books, and it displays the book titles in all uppercase characters and the length of each title string.

```
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;
import com.hp.hpl.jena.update.*;

public class Example13
{
```

```
  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szGraphName  = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
       = OracleNoSqlConnection.createInstance(szStoreName,
                                              szHostName,
                                              szHostPort);

// Create model from named graph
Model model =
      OracleModelNoSql.createOracleModelNoSql(szGraphName, conn);

// Clear model
model.removeAll();

String insertString =
" PREFIX dc: <http://purl.org/dc/elements/1.1/> "               +
" INSERT DATA "                                                 +
" { <http://example/book3> dc:title \"A new book\" ; "          +
"                          dc:creator \"A.N.Other\" . "         +
"   <http://example/book4> dc:title \"Semantic Web Rocks\" ; "  +
"                          dc:creator \"TB\" . "                +
" } ";

System.out.println("Execute insert action " + insertString);
UpdateAction.parseExecute(insertString, model);


String szQuery
= "PREFIX dc: <http://purl.org/dc/elements/1.1/> "          +
"PREFIX fn: <http://www.w3.org/2005/xpath-functions#> "   +
" SELECT ?subject (fn:upper-case(?object) as ?object1) "  +
"        (fn:string-length(?object) as ?strlen) "         +
" WHERE { ?subject dc:title ?object } ";

System.out.println("Execute query " + szQuery);

Query query = QueryFactory.create(szQuery, Syntax.syntaxARQ);
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
```

```
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example13.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example13 <store_name> \
<host_name> <host_port> <graph_name>

Execute query PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
SELECT ?subject (fn:upper-case(?object) as ?object1)
(fn:string-length(?object) as ?strlen)
WHERE { ?subject dc:title ?object }
-----------------------------------------------------------
| subject                | object1              | strlen |
===========================================================
| <http://example/book4> | "SEMANTIC WEB ROCKS" | 18     |
| <http://example/book3> | "A NEW BOOK"         | 10     |
-----------------------------------------------------------
```

Example14: SPARQL Update

This example inserts two triples into the default graph using SPARQL update.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.rdf.model.Model;
import oracle.rdf.kv.client.jena.*;
import com.hp.hpl.jena.update.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;

public class Example14
{
  public static void main(String[] args) throws Exception
```

```
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szGraphName  = args[3];

// Create Oracle NoSQL connection
OracleNoSqlConnection conn =
            OracleNoSqlConnection.createInstance(szStoreName,
                                                 szHostName,
                                                 szHostPort);

// Create model for default graph
Model model =
      OracleModelNoSql.createOracleModelNoSql(szGraphName, conn);

// Clear model
model.removeAll();

String insertString =
"PREFIX dc: <http://purl.org/dc/elements/1.1/> "       +
"INSERT DATA "                                         +
"{ <http://example/book3> dc:title \"A new book\" ; "  +
"                         dc:creator \"A.N.Other\" . " +
" } ";


System.out.println("Execute insert action " + insertString);
UpdateAction.parseExecute(insertString, model);

OracleGraphNoSql graph = (OracleGraphNoSql) model.getGraph();

// Find all triples in the default graph
ExtendedIterator<Triple> ei = GraphUtil.findAll(graph);

while (ei.hasNext()) {
      System.out.println("Triple " + ei.next().toString());
    }

ei.close();
model.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
```

```
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
\./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example14.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example14 <store_name> \
<host_name> <host_port> <graph_name>

Triple http://example/book3 @dc:creator "A.N.Other"
Triple http://example/book3 @dc:title "A new book"
```

Example15.java: Oracle NOSQL Database Connection Pooling

This example uses Oracle Database connection pooling.

```
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.query.*;
import oracle.rdf.kv.client.jena.*;

public class Example15
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
String szModelName  = args[3];
int iPoolSize = Integer.parseInt(args[4]);

// Property of the pool: wait if no connection is available
// at request.
boolean bWaitIfBusy = true;

System.out.println("Creating OracleNoSQL pool");
OracleNoSqlPool pool =
                OracleNoSqlPool.createInstance(szStoreName,
                                               szHostName,
                                               szHostPort,
                                               iPoolSize,
                                               bWaitIfBusy,
                                               true); //lazyInit

System.out.println("Done creating OracleNoSql pool");

// grab an Oracle NoSQL connection and do something
System.out.println("Get a connection from the pool");
```

```
OracleNoSqlConnection conn = pool.getResource();

OracleModelNoSql model =
                OracleModelNoSql.createOracleModelNoSql(szModelName,
                                                        conn);

System.out.println("Clear model");
model.removeAll();

model.getGraph().add(Triple.create(Node.createURI("u:John"),
                                   Node.createURI("u:cousinOf"),
                                   Node.createURI("u:Jackie")));

model.close();

//return connection back to the pool
conn.dispose();

// grab another Oracle NoSQL connection and do something
System.out.println("Get a connection from the pool");
conn = pool.getResource();
model = OracleModelNoSql.createOracleModelNoSql(szModelName, conn);
String queryString = "select ?x ?y ?z WHERE {?x ?y ?z}";

System.out.println("Execute query " + queryString);

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model);

try {
      ResultSet results = qexec.execSelect();
      ResultSetFormatter.out(System.out, results, query);
    }

finally {
      qexec.close();
    }

model.close();

//return connection back to the pool
conn.dispose();

// Close pool.
// This will close all resources even if they have not been freed up
System.out.println("Close pool, this will close all resources");
pool.close();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example15.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example15 <store_name> \
<host_name> <host_port> <graph_name>

Creating OracleNoSQL pool
Done creating OracleNoSql pool
Get a connection from the pool
Clear model
Get a connection from the pool
Execute query select ?x ?y ?z WHERE {?x ?y ?z}
---------------------------------------
| x        | y            | z          |
=======================================
| <u:John> | <u:cousinOf> | <u:Jackie> |
---------------------------------------
Close pool, this will close all resources
```

Generate Data sampling for a graph in the Oracle NoSQL Database

This example uses analyze method in the OracleGraphNoSql class to generate data sampling from a graph. In this example, data sampling is generated in a proportion sampPercentage: sampFactor with respect to all triples stored in the default graph.

```
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example16
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
double iSampRate = Double.parseDouble(args[3]);

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
```

```
          = OracleNoSqlConnection.createInstance(szStoreName,
                                                 szHostName,
                                                 szHostPort);


// Create a DatasetGraphNoSql object to manage the dataset in the
// Oracle NoSQL Database
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Clear dataset and close it as it is needed just to clear the
// dataset
datasetGraph.clearRepository();
datasetGraph.close();

// Load data from file into the Oracle NoSQL Database
DatasetGraphNoSql.load("family.rdf", Lang.RDFXML, conn,
"http://example.com");

// Analyze the default graph and gnerate sampling data
long sizeSamp = graph.analyze(iSampRate);

System.out.println("sampling size is " + sizeSamp);

graph.close();
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example16.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example15 <store_name> \
<host_name> <host_port> 0.005

sampling size is 5
```

Example16b. Generate Data sampling for the dataset in the Oracle NoSQL Database

This uses analyze method in the `OracleDatasetGraphNoSql` class to generate data sampling from the dataset. In this example, data sampling is generated using a sampling rate with respect to all triples/quads stored in the dataset.

```
import org.openjena.riot.Lang;
import oracle.rdf.kv.client.jena.*;

public class Example16b
{

  public static void main(String[] args) throws Exception
  {

String szStoreName  = args[0];
String szHostName   = args[1];
String szHostPort   = args[2];
double iSampRate = Double.parseDouble(args[3]);

// Create Oracle NoSQL connection
OracleNoSqlConnection conn
       = OracleNoSqlConnection.createInstance(szStoreName,
                                              szHostName,
                                              szHostPort);

// Create an Oracle DatasetGraphNoSql object to manage
// the dataset in the Oracle NoSQL Database
OracleGraphNoSql graph = new OracleGraphNoSql(conn);
DatasetGraphNoSql datasetGraph = DatasetGraphNoSql.createFrom(graph);

// Close graph as it is no longer needed
graph.close();

// Clear dataset and close it as it is needed just to clear the
// dataset
datasetGraph.clearRepository();

// Load data from file into the Oracle NoSQL Database
DatasetGraphNoSql.load("family.rdf", Lang.RDFXML, conn,
                           "http://example.com");

// Analyze the default graph and gnerate sampling data
long sizeSamp = datasetGraph.analyze(iSampRate);

System.out.println("sampling size is " + sizeSamp);

// Close connection
```

```
conn.dispose();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example16.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example16b <store_name>\
<host_name> <host_port> 0.005

sampling size is 5
```

Build an Ontology Model using Jena OntModel APIs

This example describes how to use OWL_MEM_RULE_INF to build an OntologyModel on top of an OracleModelNoSql instance. The inference results are added to an in-memory Jena Model.

```
import java.io.PrintStream;
import java.util.Iterator;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.graph.*;
import oracle.rdf.kv.client.jena.*;

public class ExampleOntModel
{

  public static void main(String[] szArgs) throws Exception
  {

PrintStream psOut = System.out;

psOut.println("start");
String szStoreName = szArgs[0];
String szHostName  = szArgs[1];
String szHostPort  = szArgs[2];

// Create a connection to the Oracle NoSQL Database
OracleNoSqlConnection conn
                   = OracleNoSqlConnection.createInstance(szStoreName,
                                                          szHostName,
```

```
                                                              szHostPort);

// Create an OracleGraphNoSql object to handle the default graph
// and use it to create a Jena Model object.
Node graphNode = Node.createURI("http://example.org/graph1");
OracleGraphNoSql graph = new OracleNamedGraphNoSql(graphNode, conn);
Model model =
      OracleModelNoSql.createOracleModelNoSql(graphNode, conn);

// Clear model
model.removeAll();

Node sub = Node.createURI("http://sub/a");
Node pred = Node.createURI("http://pred/a");
Node obj = Node.createURI("http://obj/a");

// Add few axioms

Triple triple = Triple.create(sub, pred, obj);
graph.add(triple);

graph.add(Triple.create(pred,
       Node.createURI("http://www.w3.org/2000/01/rdf-schema#domain"),
       Node.createURI("http://C")));

graph.add(Triple.create(pred,
   Node.createURI("http://www.w3.org/1999/02/22-rdf-syntax-ns#type"),
   Node.createURI("http://www.w3.org/2002/07/owl#ObjectProperty")));

    {
// read it out
     Iterator it = GraphUtil.findAll(graph);

     while (it.hasNext()) {
    psOut.println("triple " + it.next().toString());
    }
}

// Create an OntModel instance
OntModel om =
         ModelFactory.createOntologyModel(
                                    OntModelSpec.OWL_MEM_RULE_INF,
                                    model);

Model modelInMem = ModelFactory.createDefaultModel();
modelInMem.add(om);

    {
      Iterator it = GraphUtil.findAll(modelInMem.getGraph());
```

```
      while (it.hasNext()) {
        psOut.println("triple from OntModel " + it.next().toString());
      }
    }
model.close();
conn.close();
  }
}
```

The following are the commands to compile and run this example, as well as the expected output of the java command.

```
javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example17.java

javac -classpath ./:./jena-core-2.7.4.jar:./jena-arq-2.9.4.jar: \
./sdordfnosqlclient.jar:./kvclient.jar:./xercesImpl-2.10.0.jar: \
./slf4j-api-1.6.4.jar:./slf4j-log4j12-1.6.4.jar:./log4j/1.2.16.jar: \
./jena-iri-0.9.4.jar:./xml-apis-1.4.01.jar Example17 <store_name> \
<host_name> <host_port>

triple from OntModel http://sub/a @owl:sameAs http://sub/a
triple from OntModel http://sub/a @rdf:type rdfs:Resource
triple from OntModel http://sub/a @rdf:type owl:Thing
triple from OntModel http://sub/a @rdf:type http://C
triple from OntModel http://sub/a @http://pred/a http://obj/a
```

Chapter 9. SPARQL Gateway for XML-based Tools

SPARQL Gateway is a J2EE web application that is included with the RDF Graph feature and is designed to make semantic data easily available to applications that support XML data sources, including Oracle Business Intelligence Enterprise Edition (OBIEE).

SPARQL Gateway manages SPARQL queries and XSLT operations, executes SPARQL queries against any arbitrary standard-compliant SPARQL endpoints, and performs necessary XSL transformations before passing the response back to applications. Applications can then consume semantic data as if it is coming from an existing data source.

SPARQL Gateway Features and Benefits Overview

SPARQL Gateway handles several challenges in exposing semantic data to a non-semantic application:

- RDF syntax, SPARQL query syntax and SPARQL protocol must be understood.
- The SPARQL query response syntax must be understood.
- A transformation must convert a SPARQL query response to something that the application can consume.

To address these challenges, SPARQL Gateway manages SPARQL queries and XSLT operations, executes SPARQL queries against any arbitrary standard-compliant SPARQL endpoints, and performs necessary XSL transformations before passing the response back to applications. Applications can then consume semantic data as if it is coming from an existing data source.

Different triple stores or quad stores often have different capabilities. With the RDF Graph SPARQL Gateway, you get certain highly desirable capabilities, such as the ability to set a timeout on a long running query and the ability to get partial results from a complex query in a given amount of time. Waiting indefinitely for a query to finish is a challenge for end users, as is an application with a response time constraint. SPARQL Gateway provides both timeout and best effort query functions on top of a SPARQL endpoint. This effectively removes some uncertainty from consuming semantic data through SPARQL query executions. (See Specifying a Timeout Value (page 94) and Specifying Best Effort Query Execution (page 94) for more information.)

Installing and Configuring SPARQL Gateway

To install and configure SPARQL Gateway, follow these major steps, which are explained in subsections that follow:

1. Download the RDF Graph .zip file.
2. Deploy SPARQL Gateway in WebLogic Server.
3. Modify Proxy Settings, if necessary.
4. Configure the OracleSGDS Data Source, if necessary.

5. Add and configure the SparqlGatewayAdminGroup Group, if desired.

Download the RDF Graph .zip File

If you have not already done so, download the RDF Graph feature from My Oracle Support. For download details, see Prerequisite Software (page 106).

Note that the SPARQL Gateway Java class implementations are embedded in sdordfclient.jar.

Deploy SPARQL Gateway in WebLogic Server

Deploy the SPARQL Gateway web application (sparqlgateway.war) which is bundled in rdf_graph_for_nosql_database.zip into Oracle WebLogic Server. Verify your deployment by using your web browser to connect to the URL http://<host-name>:7001/sparqlgateway (This assumes that the Web application is deployed at port 7001).

Modify Proxy Settings

If your SPARQL Gateway is behind a firewall and you want the SPARQL Gateway to communicate with SPARQL endpoints on the internet as well as those inside the firewall, you might need to use the following JVM settings:

```
-Dhttp.proxyHost=<your_proxy_host>
-Dhttp.proxyPort=<your_proxy_port>
-Dhttp.nonProxyHosts=127.0.0.1|
<hostname_1_for_sparql_endpoint_inside_firewall>|<
hostname_2_for_sparql_endpoint_inside_firewall>|...|
<hostname_n_for_sparql_endpoint_inside_firewall>
```

You can specify these settings in the `startWebLogic.sh` script.

Add and Configure the SparqlGatewayAdminGroup Group

The following JSP files in SPARQL Gateway can help you to view, edit, and update SPARQL queries and XSL transformations that are stored in an Oracle database:

```
http://<host>:7001/sparqlgateway/admin/sparql.jsp
```

```
http://<host>:7001/sparqlgateway/admin/xslt.jsp
```

These files are protected by HTTP basic authentication. In `WEB-INF/weblogic.xml`, a principal named `SparqlGatewayAdminGroup` is defined.

To be able to log in to either of these JSP pages, you must use the WebLogic Server to add a group named `SparqlGatewayAdminGroup`, and create a new user or assign an existing user to this group.

Using SPARQL Gateway with RDF Data

The primary interface for an application to interact with SPARQL Gateway is through a URL with the following format:

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>
&<SPARQL_QUERY>&<XSLT>
```

In the preceding format:

- <SPARQL_ENDPOINT> specifies the ee parameter, which contains a URL encoded form of a SPARQL endpoint.

  For example, `ee=http%3A%2F%2Fsparql.org%2Fbooks` is the URL encoded string for SPARQL endpoint http://sparql.org/books. It means that SPARQL queries are to be executed against endpoint http://sparql.org/books.

- <SPARQL_QUERY> specifies either the SPARQL query, or the location of the SPARQL query.

  If it is feasible for an application to accept a very long URL, you can encode the whole SPARQL query and set `eq=<encoded_SPARQL_query>` in the URL. If it is not feasible for an application to accept a very long URL, you can store the SPARQL queries and make them available to SPARQL Gateway using one of the approaches described in Storing SPARQL Queries and XSL Transformations (page 92).

- <XSLT> specifies either the XSL transformation, or the location of the XSL transformation.

  If it is feasible for an application to accept a very long URL, you can encode the whole XSL transformation and set `ex=<encoded_XSLT>` in the URL. If it is not feasible for an application to accept a very long URL, you can store the XSL transformations and make them available to SPARQL Gateway using one of the approaches described in Storing SPARQL Queries and XSL Transformations (page 92).

Related topics:

- Storing SPARQL Queries and XSL Transformations (page 92).
- Specifying a Timeout Value (page 94)
- Specifying Best Effort Query Execution (page 94)
- Specifying a Content Type Other Than text/xml (page 95)

Storing SPARQL Queries and XSL Transformations

If it is not feasible for an application to accept a very long URL, you can specify the location of the SPARQL query and the XSL transformation in the <SPARQL_QUERY> and <XSLT> portions of the URL format described in Using SPARQL Gateway with RDF Data (page 91), using any of the following approaches:

- Store the SPARQL queries and XSL transformations in the SPARQL Gateway Web application itself.

  To do this, unpack the `sparqlgateway.war` file, and store the SPARQL queries and XSL transformations in the top-level directory; then pack the `sparqlgateway.war` file and redeploy it.

  The `sparqlgateway.war` file includes the following example files: `qb1.sparql` (SPARQL query) and `default.xslt` (XSL transformation).

Note

Use the file extension `.sparql` for SPARQL query files, and the file extension `.xslt` for XSL transformation files.

The syntax for specifying these files (using the provided example file names) is `wq=qb1.sparql` for a SPARQL query file and `wx=default.xslt` for an XSL transformation file.

If you want to customize the default XSL transformations, see the examples in Customizing the Default XSLT File (page 95).

If you specify `wx=noop.xslt`, XSL transformation is not performed and the SPARQL response is returned "as is" to the client.

- Store the SPARQL queries and XSL transformations in a file system directory, and make sure that the directory is accessible for the deployed SPARQL Gateway Web application.

  By default, the directory is set to `/tmp`, as shown in the following <init-param> setting:

  ```
  <init-param>
  <param-name>sparql_gateway_repository_filedir</param-name>
  <param-value>/tmp/</param-value>
  </init-param>
  ```

  It is recommended that you customize this directory before deploying the SPARQL Gateway. To change the directory setting, edit the text in between the `<param-value>` and `</param-value>` tags.

  The following examples specify a SPARQL query file and an XSL transformation file that are in the directory specified in the `<init-param>` element for `sparql_gateway_repository_filedir`:

  ```
  fq=qb1.sparql
  fx=myxslt1.xslt
  ```

- Make the SPARQL queries and XSL transformations accessible from a website.

  By default, the website directory is set to http://127.0.0.1/queries/, as shown in the following <init-param> setting:

  ```
  <init-param>
  <param-name>sparql_gateway_repository_url</param-name>
  </param-name>http://127.0.0.1/queries/</param-name>
  </init-param>
  ```

  Customize this directory before deploying the SPARQL Gateway. To change the website setting, edit the text in between the `</param-name>` and `</param-name>` tags.

  The following example specifies a SPARQL query file and an XSL transformation file that are in the URL specified in the `<init-param>` element for `sparql_gateway_repository_url`.

```
uq=qb1.sparql
ux=myxslt1.xslt
```

Internally, SPARQL Gateway computes the appropriate complete URL, fetches the content, starts query execution, and applies the XSL transformation to the query response XML.

Configure the OracleSGDS Data Source

If an Oracle database is used for storage of and access to SPARQL queries and XSL transformations for SPARQL Gateway, then you must configure a data source named `OracleSGDS`.

To create this data source, follow the instructions in Use Oracle WebLogic Server (page 8); however, specify OracleSGDS as the data source name instead of OracleSemDS.

If the OracleSGDS data source is configured and available, SPARQL Gateway servlet will automatically create all the necessary tables and indexes upon initialization.

Specifying a Timeout Value

When you submit a potentially long-running query using the URL format described in Using SPARQL Gateway with RDF Data (page 91), you can limit the execution time by specifying a timeout value in milliseconds. For example, the following shows the URL format and a timeout specification that the SPARQL query execution started from SPARQL Gateway is to be ended after 1000 milliseconds (1 second):

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&<
SPARQL_QUERY>&<XSLT>&t=1000
```

If a query does not finish when timeout occurs, then an empty SPARQL response is constructed by SPARQL Gateway.

Note that even if SPARQL Gateway times out a query execution at the HTTP connection level, the query may still be running on the server side. The actual behavior will be vendor-dependent.

Specifying Best Effort Query Execution

Note

You can specify best effort query execution only if you also specify a timeout value (described in the previous section, Specifying a Timeout Value (page 94)).

When you submit a potentially long-running query using the URL format described in Using SPARQL Gateway with RDF Data (page 91), if you specify a timeout value, you can also specify a "best effort" limitation on the query. For example, the following shows the URL format with a timeout specification of 1000 milliseconds (1 second) and a best effort specification (&b=t):

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&
<SPARQL_QUERY>&<XSLT>&t=1000&b=t
```

The `web.xml` file includes two parameter settings that affect the behavior of the best effort option: `sparql_gateway_besteffort_maxrounds` and `sparql_gateway_besteffort_maxthreads`. The following show the default definitions:

```
<init-param>
<param-name>sparql_gateway_besteffort_maxrounds</param-name>
</param-name>10</param-name>
</init-param>
```

```
<init-param>
<param-name>sparql_gateway_besteffort_maxthreads</param-name>
</param-name>3</param-name>
</init-param>
```

When a SPARQL SELECT query is executed in best effort style, a series of queries will be executed with an increasing LIMIT value setting in the SPARQL query body. (The core idea is based on the observation that a SPARQL query runs faster with a smaller LIMIT setting.) SPARQL Gateway starts query execution with a "LIMIT 1" setting. Ideally, this query can finish before the timeout is due. Assume that is the case, the next query will have its LIMIT setting increased, and subsequent queries have higher limits. The maximum number of query executions is controlled by the `sparql_gateway_besteffort_maxrounds` parameter.

If it is possible to run the series of queries in parallel, the `sparql_gateway_besteffort_maxthreads` parameter controls the degree of parallelism.

Specifying a Content Type Other Than text/xml

By default, SPARQL Gateway assumes that XSL transformations generate XML, and so the default content type set for HTTP response is text/xml. However, if your application requires a response format other than XML, you can specify the format in an additional URL parameter (with syntax &rt=), using the following format:

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&
<SPARQL_QUERY>&<XSLT>&rt=<content_type>
```

Note that <content_type> must be URL encoded.

Customizing the Default XSLT File

You can customize the default XSL transformation file (the one referenced using `wx=default.xslt`). This section presents some examples of customizations.

The following example implements this namespace prefix replacement logic: if a variable binding returns a URI that starts with http://purl.org/goodrelations/v1#, that portion is replaced by gr:; and if a variable binding returns a URI that starts with http://www.w3.org/2000/01/rdf-schema#, that portion is replaced by rdfs:.

```
<xsl:when test="starts-with(text(),
'http://purl.org/goodrelations/v1#')">
<xsl:value-of select="concat('gr:',substring-after(text(),
'http://purl.org/goodrelations/v1#'))"/>
```

```
</xsl:when>
...
<xsl:when test="starts-with(text(),
                          'http://www.w3.org/2000/01/rdf-schema#')">
   <xsl:value-of select="concat('rdfs:',substring-after(text(),
'http://www.w3.org/2000/01/rdf-schema#'))"/>
</xsl:when>
The following example implements logic to trim a leading http://localhost/
or a leading http://127.0.0.1/.
<xsl:when test="starts-with(text(),'http://localhost/')">
  <xsl:value-of select="substring-after(text(),'http://localhost/')"/>
</xsl:when>
<xsl:when test="starts-with(text(),'http://127.0.0.1/')">
  <xsl:value-of select="substring-after(text(),'http://127.0.0.1/')"/>
</xsl:when>
```

Using the SPARQL Gateway Graphical Web Interface

SPARQL Gateway provides several browser-based interfaces to help you test queries, navigate semantic data, and manage SPQARQL query and XSLT files:

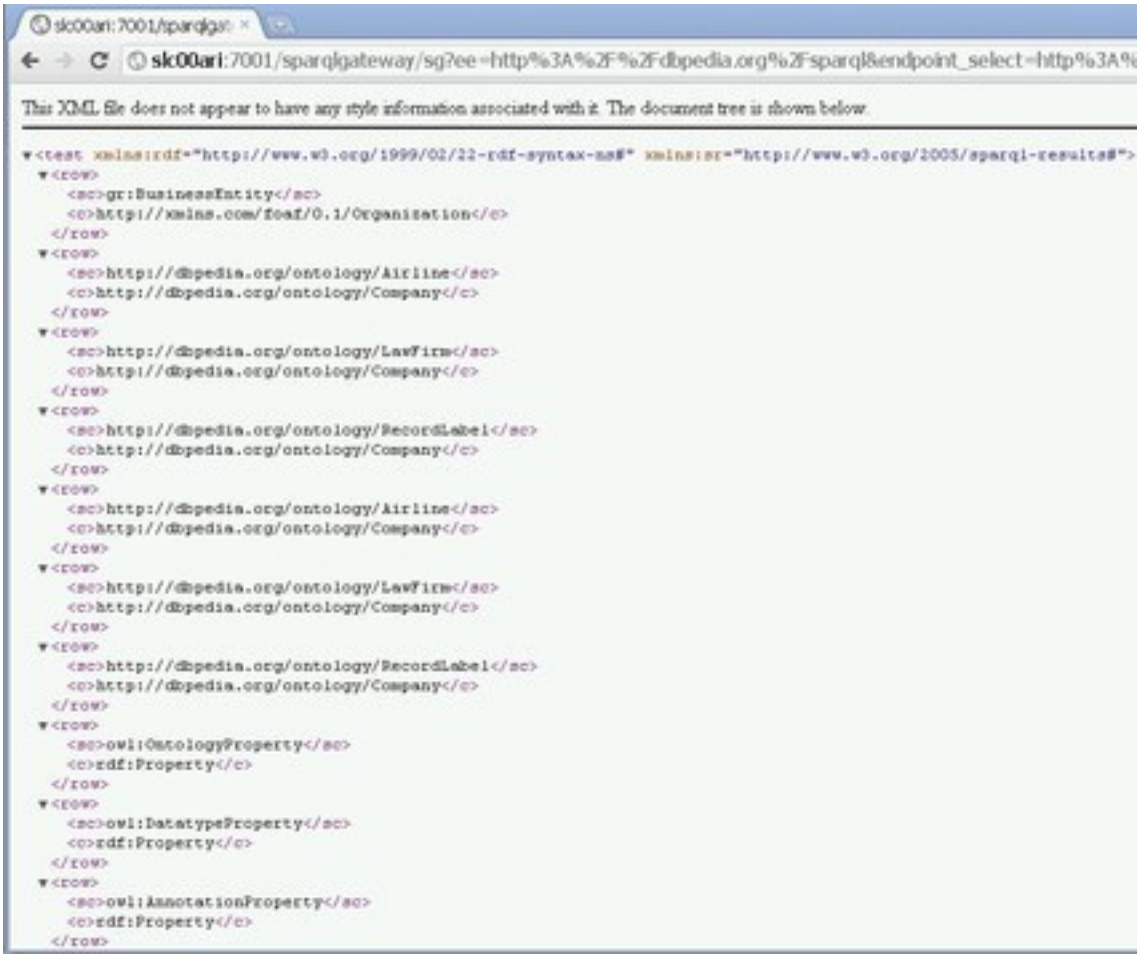
- Main Page (index.html) (page 96)
- Navigation and Browsing Page (browse.jsp) (page 98)
- XSLT Management Page (xslt.jsp) (page 100)
- SPARQL Management Page (sparql.jsp) (page 101)

Main Page (index.html)

http://<host>:<port>/sparqlgateway/index.html provides a simple interface for executing SPARQL queries and then applying the transformations in the default.xslt file to the response. The following shows this interface for executing a query.

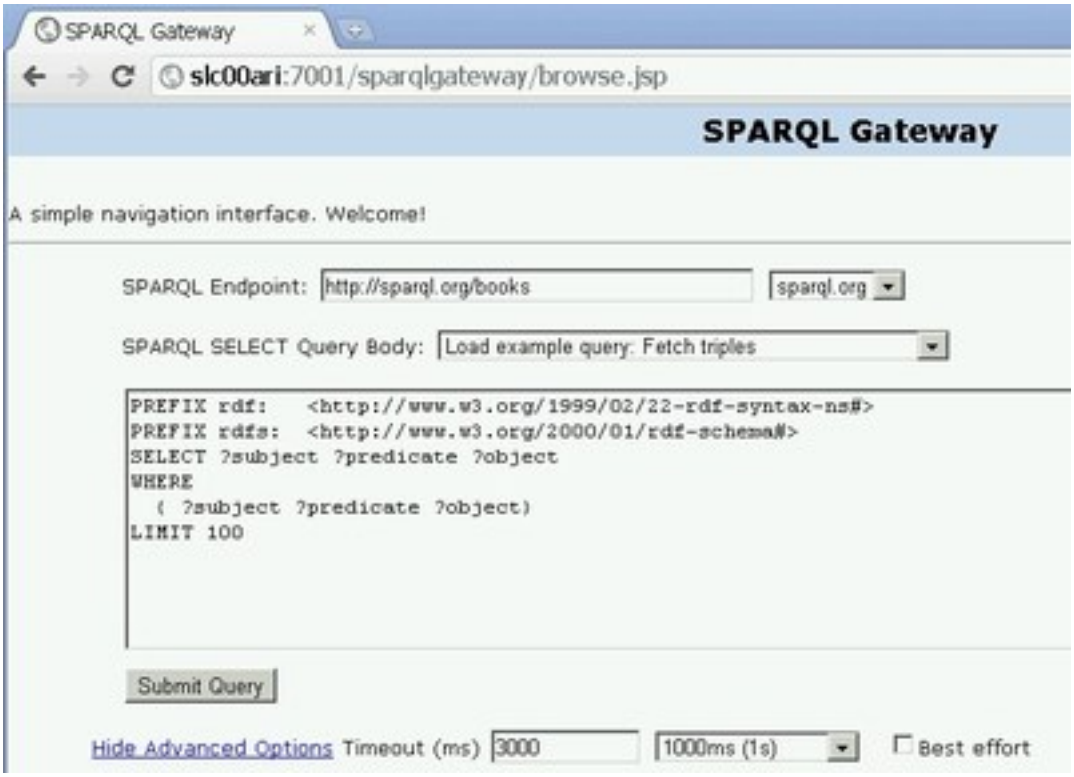
Enter or select an endpoint, specify the `SPARQL SELECT Query Body`, and press `Submit Query`.

For example, if you specify http://dbpedia.org/sparql as the SPARQL endpoint and use the SPARQL query body shown in the previous figure, the response will be similar to that shown in the next figure. Note that the default transformations (in default.xslt) have been applied to the XML output in this figure.



Navigation and Browsing Page (browse.jsp)

http://<host>:<port>/sparqlgateway/index.html provides a simple interface for executing SPARQL queries and then applying the transformations in the default.xslt file to the response. The following figure shows this interface for executing a query.

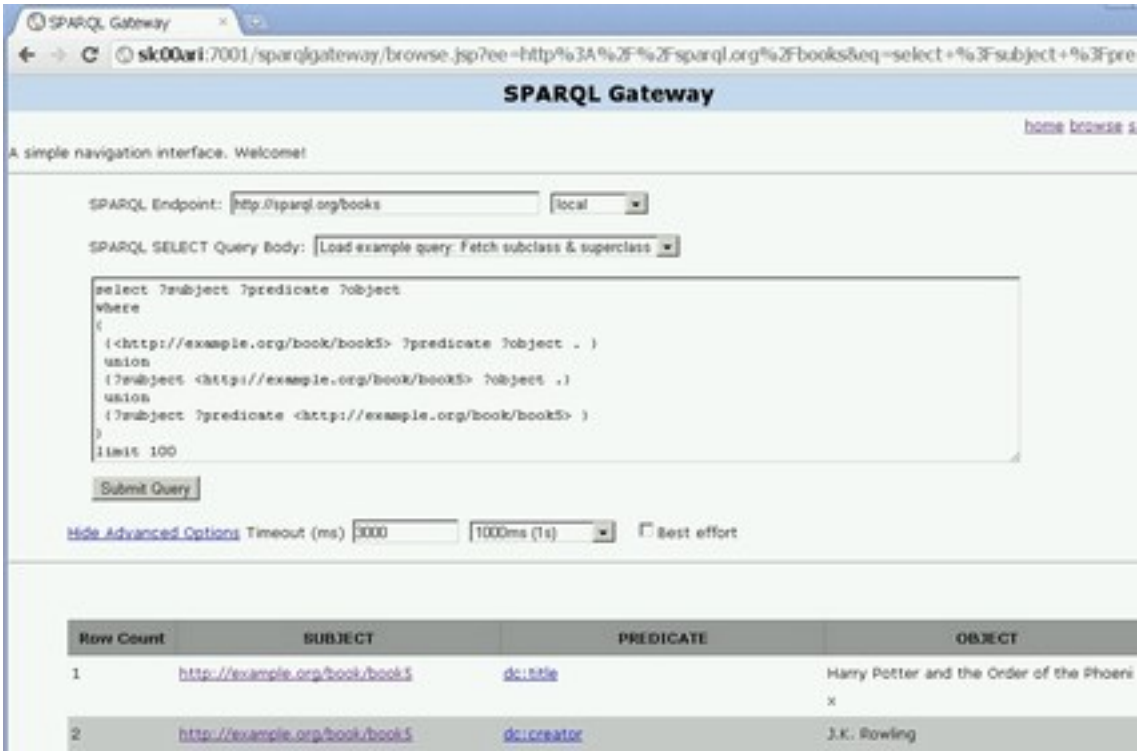


Enter or select a `SPARQL Endpoint`, specify the `SPARQL SELECT Query Body`, optionally specify a `Timeout (ms)` value and the `Best Effort` option, and press `Submit Query`.

The SPARQL response is parsed and then presented in table form, as shown in the following figure:

In the previous figure, note that URIs are clickable to allow navigation, and that when users move the cursor over a URI, tool tips are shown for the URIs which have been shortened for readability (as in http://purl.org.dc/elements/1.1/title being displayed as the tool tip for dc:title in the figure).

If you click the URI http://example.org/book/book5 in the output shown above, a new SPARQL query is automatically generated and executed. This generated SPARQL query has three query patterns that use this particular URI as subject, predicate, and object, as shown in the next figure. Such a query can give you a good idea about how this URI is used and how it is related to other resources in the data set.

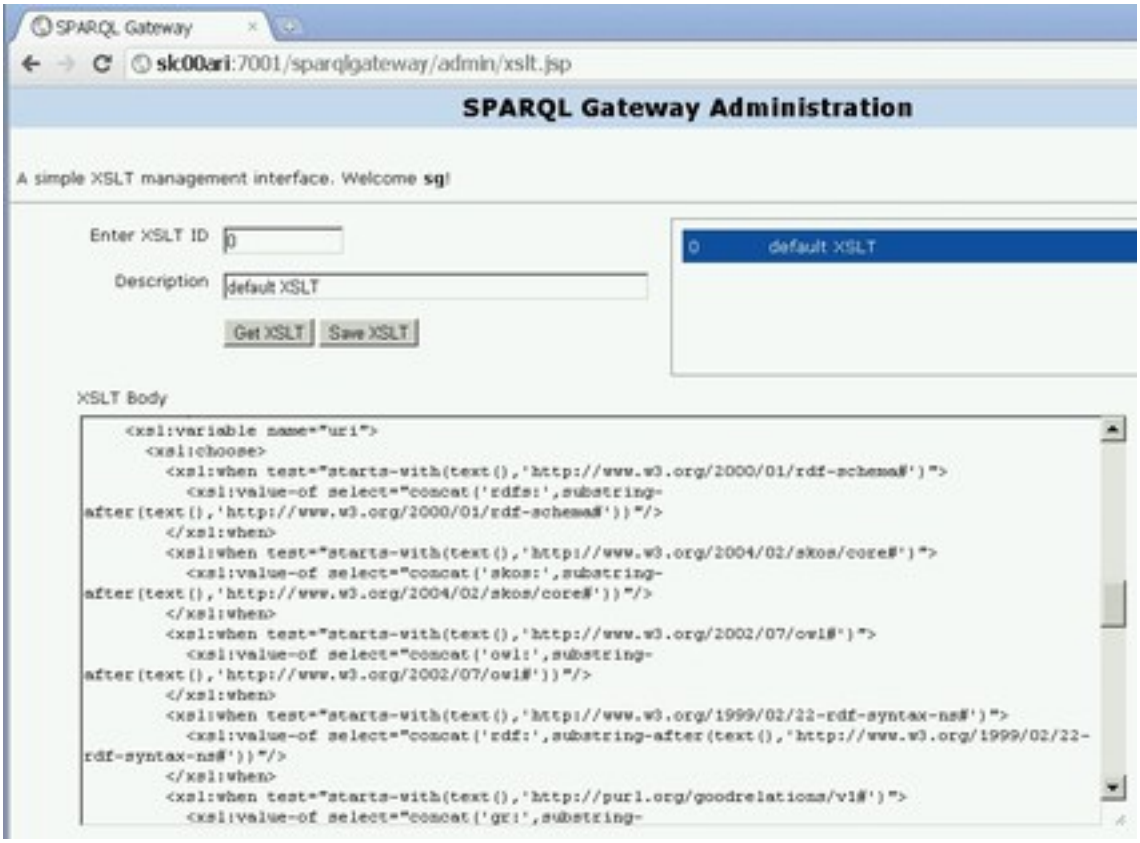


When there are many matches of a query, the results are organized in pages and you can click on any page. The page size by default is 50 results. To display more (or fewer) than 50 rows per page in a response with the Browsing and Navigation Page (browse.jsp), you can specify the &resultsPerPage parameter in the URL. For example, to allow 100 rows per page, include the following in the URL:

```
&resultsPerPage=100
```

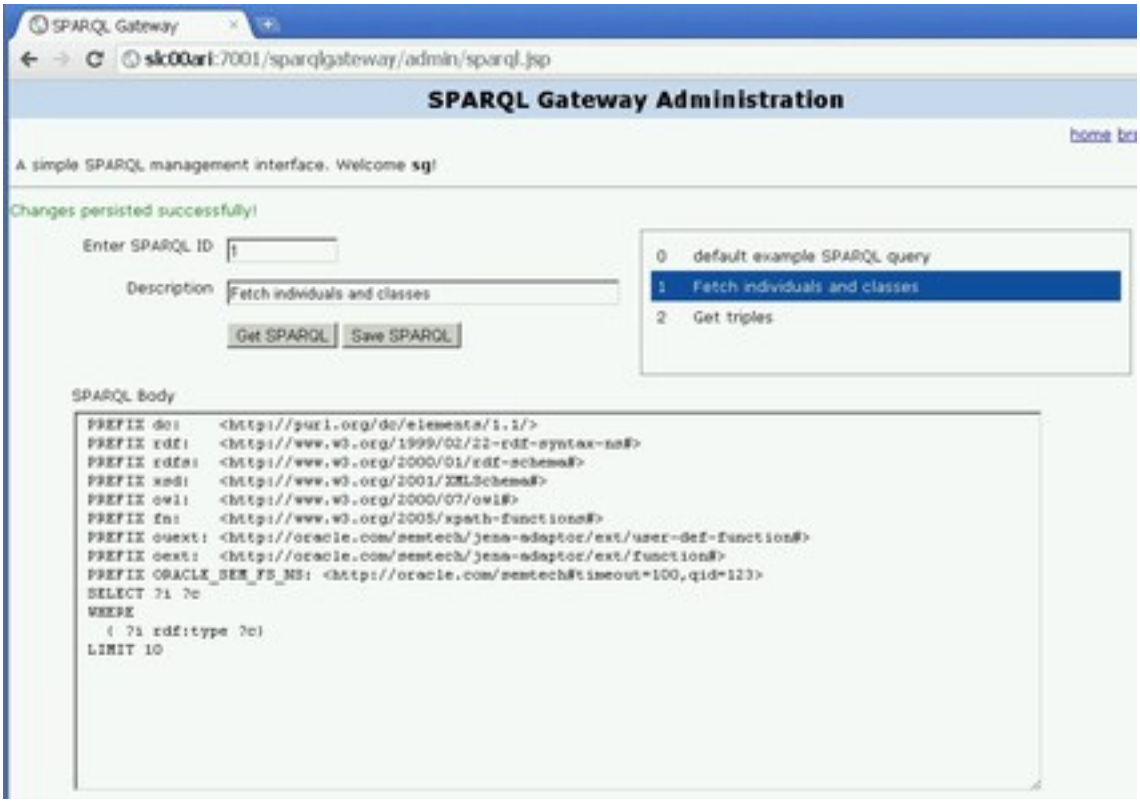
XSLT Management Page (xslt.jsp)

http://<host>:<port>/sparqlgateway/admin/xslt.jsp provides a simple XSLT management interface. You can enter an XSLT ID (integer) and click `Get XSLT` to retrieve both the Description and XSLT Body. You can modify the XSLT Body text and then save the changes by clicking `Save XSLT`. Note that there is a previewer to help you navigate among available XSLT definitions.



SPARQL Management Page (sparql.jsp)

http://<host>:<port>/sparqlgateway/admin/xslt.jsp provides a simple SPARQL management interface. You can enter a SPARQL ID (integer) and click `Get SPARQL` to retrieve both the Description and SPARQL Body. You can modify the SPARQL Body text and then save the changes by clicking `Save SPARQL`. Note that there is a previewer to help you navigate among available SPARQL queries.

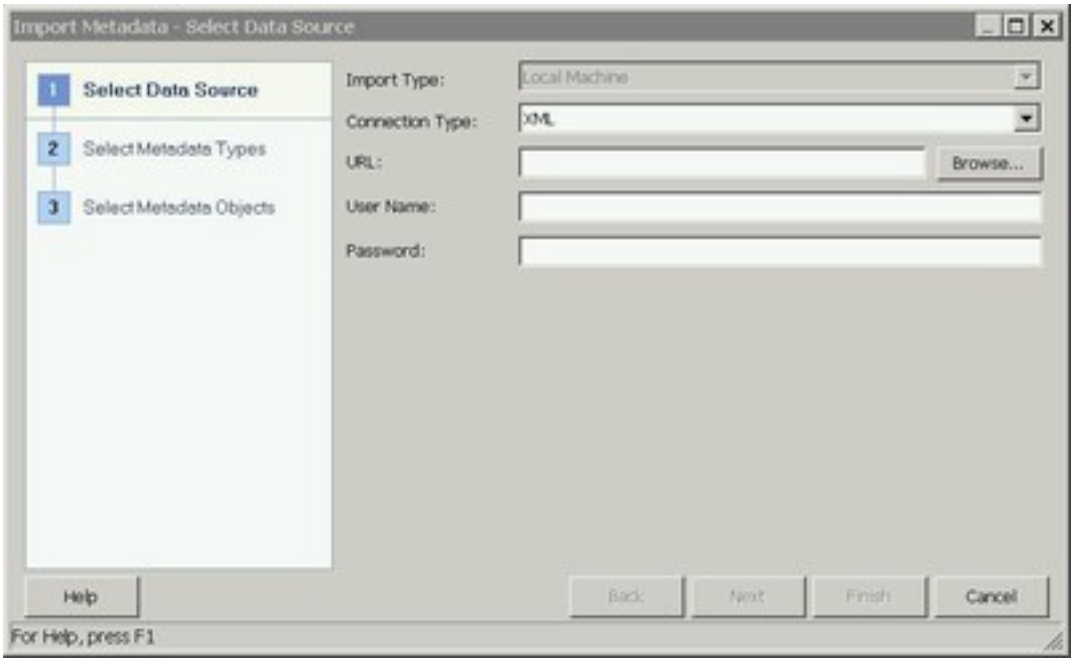


Using SPARQL Gateway as an XML Data Source to OBIEE

This section explains how to create an XML Data source for Oracle Business Intelligence Enterprise Edition (OBIEE), by integrating OBIEE with RDF using SPARQL Gateway as a bridge. (The specific steps and illustrations reflect the Oracle BI Administration Tool Version 11.1.1.3.0.100806.0408.000.)

1. Start the Oracle BI Administration Tool.

2. Click File, then Import Metadata. The first page of the Import Metadata wizard is displayed, as shown in the following figure:

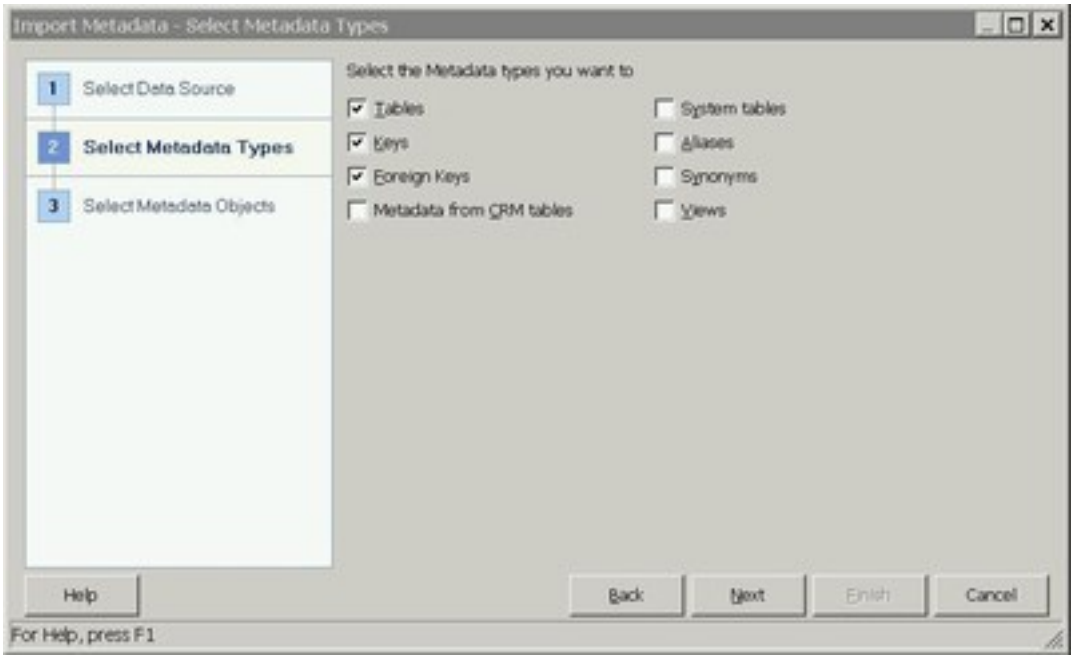


Connection Type: Select XML.

URL: This is the URL required for an application to interact with SPARQL Gateway, as explained in Using SPARQL Gateway with RDF Data (page 91). You can also include the timeout and best effort options.

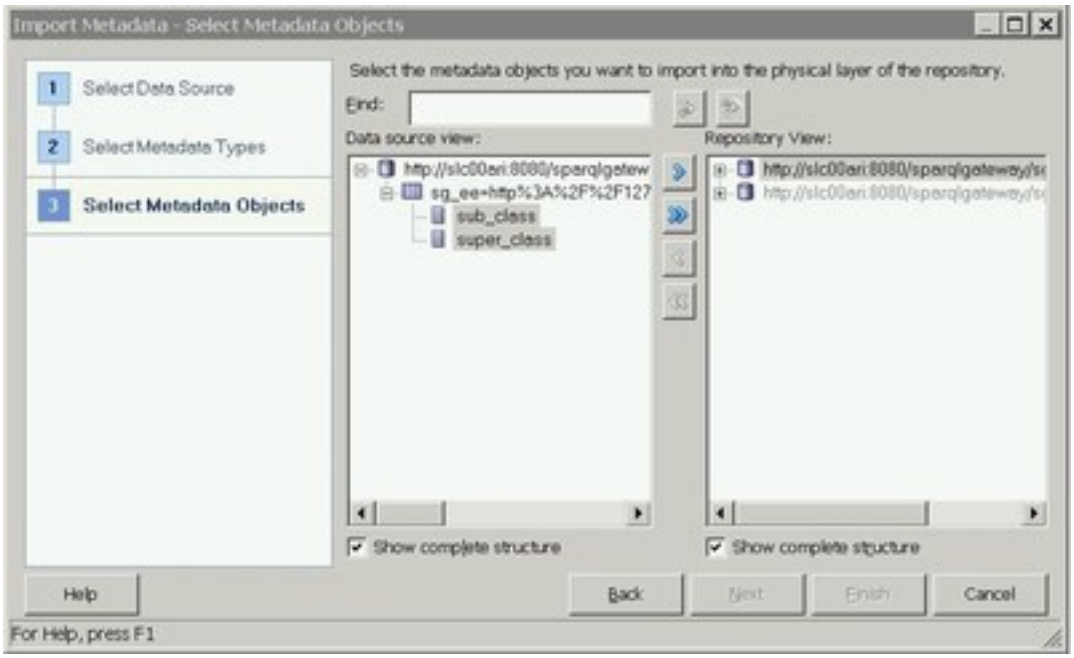
Ignore the `User Name` and `Password` fields.

3. Click `Next`. The second page of the Import Metadata wizard is displayed, as shown in the following figure:



Select the desired metadata types to be imported. Be sure that `Tables` is included in the selected types.

4. Click `Next`. The third page of the Import Metadata wizard is displayed, as shown in the following figure:



In the Data Source View, expand the node that has the table icon, select the column names (mapped from projected variables defined in the SPARQL SELECT statement), and click the right-arrow (>) button to move the selected columns to the Repository View.

5. Click `Finish`.

6. Complete the remaining steps for the usual OBIEE Business Model work and Mapping and Presentation definition work, which are not specific to SPARQL Gateway or RDF data.

Appendix A. Prerequisite Software

To use the RDF Graph feature, you must first ensure that the system environment has the necessary software, including Oracle NoSQL Database Enterprise Edition 2.0.23, Apache Jena version 2.7.4, the RDF Graph feature, and JDK 1.6. (Update 25 or later). You can also manually install these components if need be:

- Oracle NoSQL Database Enterprise Edition 2.0.23. Further details on this installation can be found on http://www.oracle.com/technetwork/products/nosqldb/downloads/index.html
- Apache Jena (version 2.7.4), Apache Jena ARQ (version 2.9.4), and Apache Jena Joseki (version 3.4.4) are included with the RDF Graph feature. They can also be downloaded from http://archive.apache.org/dist/jena/binaries/apache-jena-2.7.4.zip (The directory or folder into which it is unzipped is referred to as <Jena_DIR>.)
- The RDF Graph feature can be downloaded from the Oracle Software Delivery Cloud located at https://edelivery.oracle.com:

  1. Choose `Oracle Database` as the product pack and `Generic Platform` as the platform.
  2. Click the `Go` button.
  3. Select the `Oracle NoSQL Database Media Pack for Generic Platform`.
  4. Click the `Continue` button.
  5. Click the `Download` button for "RDF Graph for Oracle NoSQL Database Enterprise Edition."

Appendix B. Generating a New SPARQL Service WAR file

It is possible to modify the SPARQL endpoint Web Application Archive (WAR file), for instance, to enhance its logic by adding a servlet or a filter. Generating a new SPARQL endpoint WAR file involves downloading Apache Jena Joseki, an open source SPARQL server that supports the SPARQL protocol and SPARQL queries. This section explains how to generate a web application archive (joseki.war) for the SPARQL Service endpoint.

1. Ensure that you have Java 6 installed, because it is required by Joseki 3.4.4.
2. Download Apache Jena Joseki 3.4.4 (joseki-3.4.4.zip) from http://sourceforge.net/projects/joseki/files/Joseki-SPARQL/.
3. Unpack joseki-3.4.4.zip into a temporary directory. For example:

   ```
   mkdir /tmp/joseki
   cp joseki-3.4.4.zip /tmp/joseki
   cd /tmp/joseki
   unzip joseki-3.4.4.zip
   ```

4. Ensure that you have downloaded and unzipped the RDF Graph feature for the Oracle NoSQL Database, as explained in Setup the System Environment (page 5).
5. Create a directory named `joseki.war` at the same level as the jena_adapter directory, and go to it. For example:

   ```
   mkdir /tmp/joseki.war
   cd /tmp/joseki.war
   ```

6. Copy necessary files into the directory created in the preceding step:

   ```
   cp /tmp/jena_adapter/joseki/* /tmp/joseki.war
   cp -rf /tmp/joseki/Joseki-3.4.4/webapps/joseki/StyleSheets \
   /tmp/joseki.war
   ```

7. Create directories and copy necessary files into them, as follows:

   ```
   mkdir /tmp/joseki.war/WEB-INF
   cp /tmp/jena_adapter/web/* /tmp/joseki.war/WEB-INF

   mkdir /tmp/joseki.war/WEB-INF/lib
   cp /tmp/joseki/Joseki-3.4.4/lib/joseki-3.4.4.jar \
   /tmp/joseki.war/WEB-INF/lib
   cp /tmp/jena_adapter/jar/*.jar /tmp/joseki.war/WEB-INF/lib
   cp <#JENA_DIR>/lib/* /tmp/joseki.war/WEB-INF/lib

   ## Assume KV_HOME points to the home directory of an
   ## Oracle NoSQL Database
   ## Release <#ORACLE>.
   ```

```
cp $KVHOME/lib/kvclient.jar /tmp/joseki.war/WEB-INF/lib
```

8. Modify Apache Jena Joseki's configuration file (`joseki-config.ttl`) file located in `/tmp/joseki.war` to specify the store name, host name, and host port to access the Oracle NoSQL Database. This data will be used by the SPARQL Service endpoint to establish connections to the Oracle NoSQL Database and execute update and query operations. For detailed information about this configuration, see Configuring an Oracle NoSQL Database connection in the SPARQL service (page 10).

9. Check the files and the directory structure to make sure they reflect the following:

```
.
|-- META-INF
| |-- MANIFEST.MF
|-- StyleSheets
| |-- joseki.css
|-- WEB-INF
| |-- lib
| | |-- common-codec-1.5.jar
| | |-- httpclient-4.1.2.jar
| | |-- httpcore-4.1.3.jar
| | |-- jena-arq-2.9.4.jar
| | |-- jena-core-2.7.4.jar
| | |-- jena-iri-0.9.4.jar
| | |-- jena-tdb-0.9.4.jar
| | |-- joseki-3.4.4.jar
| | |-- kvclient.jar
| | |-- log4j-1.2.16.jar
| | |-- sdordfnosqlclient.jar
| | |-- slf4j-api-1.6.4.jar
| | |-- slf4j-log4j12-1.6.4.jar
| | |-- xercesImpl-2.10.0.jar
| | |-- xml-apis-1.4.01.jar
| |-- web.xml
|-- application.xml
|-- index.html
|-- joseki-config.ttl
|-- update.html
|-- xml-to-html.xsl
```

10. Build a .war file from the `/tmp/joseki.war` directory (a .war file is required if you want to deploy Apache Jena Joseki to an OC4J container), using the following commands:

```
cd /tmp/joseki.war
jar cvf /tmp/joseki_app.war *
```

Appendix C. Third Party Licenses

This appendix contains license notices for third-party products included with RDF Graph for Oracle NoSQL Database.

Oracle acknowledges that the following third-party proprietary and open source software are used in the provided programs covered by this documentation.

Apache Jena Apache License

Version 2.0, January 2004

http://www.apache.org/licenses

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions.**

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain

separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

    a. You must give any other recipients of the Work or Derivative Works a copy of this License; and

    b. You must cause any modified files to carry prominent notices stating that You changed the files; and

    c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this

License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

ICU License

https://github.com/OpenRefine/OpenRefine/blob/master/licenses/icu4j.LICENSE.txt

ICU License – ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2009 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Licensing terms for SLF4J

http://www.slf4j.org/license.html

Copyright © 2004-2013 QOS.ch

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without

restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

These terms are identical to those of the MIT License, also called the X License or the X11 License, which is a simple, permissive non-copyleft free software license. It is deemed compatible with virtually all types of licenses, commercial or otherwise. In particular, the Free Software Foundation has declared it compatible with GNU GPL. It is also known to be approved by the Apache Software Foundation as compatible with Apache Software License.

Protégé-OWL

Protégé-OWL is available as free software under the open-source Mozilla Public License.

http://www.mozilla.org/MPL/2.0/

Mozilla Public License

Version 2.0

1. Definitions

1.1. "Contributor"

means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

1.2 "Contributor Version"

means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor's Contribution.

1.3 "Contribution"

means Covered Software of a particular Contributor.

1.4 "Covered Software"

means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.

1.5 "Incompatible With Secondary Licenses"

means

a. that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or

b. that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.

1.6 "Executable Form"

means any form of the work other than Source Code Form.

1.7 "Larger Work"

means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8 "License"

means this document.

1.9 "Licensable"

means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10 "Modifications"

means any of the following:

a. any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or

b. any new file in Source Code Form that contains any Covered Software.

1.11 "Patent Claims" of a Contributor

means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12 "Secondary License"

means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13 "Source Code Form"

means the form of the work preferred for making modifications.

1.14 "You" (or "Your")

means an individual or a legal entity exercising rights under this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions

2.1 Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

a. under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and

b. under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.

2.2 Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3 Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

a. for any code that a Contributor has removed from Covered Software; or

b. for infringements caused by: (i) Your and any other third party's modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or

c. under Patent Claims infringed by Covered Software in the absence of its Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4 Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5 Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6 Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7 Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities

3.1 Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2 Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

a. such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

b. You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3 Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4 Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source

Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5 Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty

Covered Software is provided under this License on an "as is" basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-

infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability

Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such party's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the

license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at http://mozilla.org/MPL/2.0/. If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - "Incompatible With Secondary Licenses" Notice

This Source Code Form is "Incompatible With Secondary Licenses", as defined by the Mozilla Public License, v. 2.0.