Verification of the RDF implementation in DB2 10.1 for Linux

Yuqing He

Thesis

Business Information Technology

10, May 2013

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation

Degree programme

Author or authors Yuqing He	**Group or year of entry** BiTE 2009
Title of the thesis Verification of the RDF implementation in DB2 10.1 for Linux	**Number of pages and appendices** 50 + 18
Teacher(s) or supervisor(s) Martti Laiho, Arvo Lipitsainen	

This study is a proof of concept of the RDF the implementation of in DB2 Express-C version 10.1. The study is based on the IBM Developerworks tutorial: "Resource description framework application development in DB2 for Linux, UNIX, and Windows". Two research questions have been answered in this study:

1. How to implement an RDF application in DB2 10.1 for Linux?
2. What was difficult in the implementaion, problems and solutions?

The implementation was conducted in a Linux operation system, which is different from the Windows operation system that was used in the tutorial. During the implemention, we encountered several problems. These problems and corresponding solutions are recorded in the study. The results of this study prove that the concept presented in the tutorial is positive. Loading data into an RDF is possible, updating of the data in the RDF store works. In addition, one can also administrate the RDF store through two different methods: automatic statistics collection and manual statistics collection.

Throughout the study, we have found some improvements. The sample store that we used in this tutorial has no unique constraint in the database. The store allows us to insert the same dataset multiple times which would result in a large dataset which contains the repeated content. For example, employee id = 00001 as well as his information can be recorded multiple times in the database. We suggest further study of RDF constraints in future.

Keywords
RDF, DB2, Semantic web, NoSQL, Database, SPARQL

Table of contents

List of Figures:

Table:

List of Screenshots

Glossary of Terms

RDF	Resources Description Framework
SPARQL	RDF query language for database
DB2	Relational model database server developed by IBM
SOA	Service-oriented Architecture
SQL	Structured Query Language
OS	Operating System
RDBMS	Relational database management system
XML	Extensible Mark-up Language
ARQ	A SPARQL Processor for Jena
JENA	Java framework for building Semantic Web application
OWL	Web-Ontology Language
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
OVA	Open Virtualization Format
URI	Uniform Resource Identifier
RDFS	RDF Schema
OpenSUSE	A general-purpose operating system built on top of the Linux kernel

1 Introduction

With the development of the Internet, the penetration rate of using the Internet is getting higher and higher. People use and exchange a large number of data every day around the world. We can see our photographs on the web, we can see our appointments in a calendar, and now we can even see our pictures in a calendar enabling us to see what we were doing at the time we took the picture. This is due to the functionality of the Semantic Web, which is a web of data that allows this to happen.

Traditionally, data is controlled by applications and each application keeps the data to itself. Thus the data cannot be reorganized, neither reused across applications. Semantic Web differs from the traditional way of data management. According to the W3C (World Wide Web Consortium 2013): "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries." In 2005, IBM published an article named: The future of the Web is Semantic. (Balani 2005) The article stated as a conclusion that Semantic Web is an efficient way to represent data, over the World Wide Web. With Semantic Web technology, organizations may provide a unified view of data across their applications, which allows for precise retrieval of information. Since the data can be reused over the web, it simplifies enterprise and SOA (Service-oriented architecture) integration, reduces data redundancy, and provides uniform semantic meaning across applications. Since then, IBM and other technology & consulting corporations have been actively participating in research and development of Semantic Web technologies. In order to understand Semantic Web, it is necessary to understand Resource Description Framework (RDF) also, as RDF is a standard model for data interchange on the Web, and Semantic Web is based on this framework.

"RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed." (RDF Working Group 2004) A detailed description of RDF and the functionality of this technology will be further reviewed in Chapter 3

where the background theories are presented. This thesis study uses "Resource description framework application development in DB2 for Linux, UNIX, and Windows" a tutorial published by IBM. We will build a simulation model, examine the model and verify the model provided by the tutorial.

The target audience of this paper is enthusiasts and professionals who are interested in data development. The paper offers an overall knowledge on RDF, especially focusing on building a sample RDF application on DB2. For others who are interested in the IBM tutorial: RDF application development in DB2, the forth chapter of the paper, could be used as a troubleshooting and solution guide book for the application implementation. A basic knowledge of data development, DB2 and Linux of the reader is a plus to understand and better use this paper. However, one can always learn something from reading regardless the extent of knowing the subject.

The starting point of this thesis study is gathering information on the research topic: Semantic Web, Resource Description Framework and read through the IBM tutorial. In addition, the related knowledge such as SPARQL, DB2, Linux, and Java have also been studied and reviewed. We used a learn-as-you-go method of implementation and research. Several difficulties occurred during the implementation. In Chapter 4, these problems and the corresponding solutions will be provided. In the same chapter, additional notes are also placed for beginners, who are not familiar with the subject. This includes the unclear db2 command and Linux command, which were not indicated in the tutorial. Since we took Linux SUSE, instead of Windows as our operation system, some Linux commands are different from the given command in the IBM tutorial, because the tutorial was written based on the Windows operation system. These changes will also be included in the Chapter 4.

The research problem was twofold and consisted of two questions. The first question is "How to implement RDF application development on DB2?" and it aimed to describe what had been done on the system to make the application work. The second question is "what was difficult in the implementation, problems and what solutions have we found to these problems?" By answering this question, we verify the imple-

mentation as well as the tutorial. The result of the study will declare what has worked and what has not worked successfully on the system, and what can be improved for the future development on RDF in DB2. The aim of the verification on the experiment is to go over the tutorial and verify the concept of RDF application development in DB2, Linux operation system, is correct and the expected result is received. In case of the scope and limitation, this study was developed and tested on DB2 Linux environment only; more detailed information of the research is placed in the next chapter, research methodology.

The last chapter of this paper is reserved for conclusion. In this chapter, the study result will be presented. Furthermore, future development on the application will be suggested based on the implementation, and a short review on the entire study will be given to sum up the study result.

2 Research Methodology

2.1 Scope and limitation of the study

A crucial step of the thesis process is to state in a clear manner the scope and limitations of the study. The scope of this thesis is based on the research method, proof of concept, to demonstrate the tutorial and verify the process provided in the tutorial. It is important for the reader to understand that this study is solely based on the material: Resource description framework application development in DB2 10 for Linux, UNIX, and Windows part 1 and part 2. It does not introduce any new concept or functions other than those found in the tutorial. The research of this study is based on DB2 platform only; Oracle and other development platforms are not involved. Furthermore, the implementation of the RDF application development on DB2 is processed in a Linux based operation system, OpenSUSE, thus some of the command lines are slightly different from the ones that are indicated in the tutorial, which is based on the Windows operation system.

2.2 Research questions

Two research questions will be answered with evaluation in the paper:

1. How to implement RDF application development on DB2 10.1 for Linux?
2. What was difficult in the implementation, problems and solutions?

The first question aims to map out the implementation of RDF application on DB2 platform and its usability. It should explain what has been done and what has been done differently from the tutorial to make the system. In addition, some changes have been made during the implementation, in order to achieve the same result of the tutorial. These changes are recorded and will be presented in the Chapter 4. However, this paper is not going to repeat the every detailed step of the implementation, since this is not a copy pasted work of the IBM tutorial. The second question is verification on the study result. The aim of this study is to understand what RDF is and how does the RDF application work on a database by studying the IBM tutorial and implementing

this sample RDF application on DB2. Furthermore, our aim is to verify this tutorial together based on our experiment on the system. To answer the second question, we first will present the study result based on the implementation, and this is going to be placed in Chapter 5. Meanwhile, we prove what has worked during the implementation, what has not worked, what has been found difficult, and how can we improve the implementation process. Second, due to time limit, the unsolved problems will be listed in the "future development" at the end of the chapter.

Tarvainen defines that research is both mapping and explanatory in nature. It is both aiming to find new viewpoints and describe a target, as well as to find an explanation and solution to the problem. (H 2003) In order to get valid results and find solutions to the research questions, we have made several interviews with IBM RDF and DB2 related experts, as well as reading a large number of reliable materials.

2.3 Research method

As mentioned earlier, the aim of the study is to experiment the implementation of an RDF application in DB2 10.1 for Linux and verify the result. The concept of the application development comes from the IBM tutorial: Resource description framework application development in DB2 10 for Linux, UNIX, and Windows. The tutorial provides an instruction to building and maintaining a sample RDF application. The research method of this study is practical. For the purpose of the study, we have downloaded a virtual image of DB2 10 for Linux (32-bit) from DB2 on Campus website, which is a program supported by IBM that collaborates between students, faculty and professional communities. (DB2onCampus 2013) This virtual image was prepared for VMware and was imported to an Oracle Virtualbox on a Mac computer of the author, because at Haaga-Helia University of Applied Sciences the Oracle VirtualBox was the common software used amongs such projects. This research adopts the principle of proof of concept. It starts first with the implementation of the RDF application on DB2. Meanwhile, it examines the experiment and verifies the tutorial. The expected result is to prove that RDF application is able to run on DB2 10.1 in a 32-bit Linux operating system.

3 Used Technologies

In this study, we have installed VirtualBox on a Mac OS laptop to run Linux SUSE operation system with DB2 10 installed. Similar to VMware, VirtualBox is a cross-platform virtualization application, which installs on your existing Intel or AMD-based computers to run multiple operation systems at the same time. (Virtualbox.org 2013) Several technologies have been covered and applied throughout this study. Before one starts the implementation and research analysis, it is important to present the involved technology and software in a clear sense. A list of involved technologies and software are as follows:

table 3-1 involved technologies and software

Technologies	Software
SPARQL	ARQ 2.8.5 JOSEKI
RDF	IBM Data Studio (Eclipse) JENA 2.6.3
DB2	DB2 Express-C, Version 10.1
Operating Platform	Linux Open SUSE VirtualBox 4.2.6 or higher

Additionally, the table above represents the RDF architecture as well. The bottom of the table presents the lowest base, a VirtualBox that we have installed on the computer to run the virtual environment. Into this virtual environment, we imported the OVA

file appliance from IBM containing Linux Open SUSE as our operating system to run Linux on a Mac OS based computer. The readily installed DB2 Express-C V10.1 is the platform where we implement RDF application. And in RDF, SPARQL is the query language to make queries into an RDF database.

3.1 What is resource description framework (RDF)?

Resource Description Framework (RDF) is a framework for describing resources on the web, encoding metadata and other knowledge on the semantic web. According to w3school's information (w3schools, 2013), RDF is been used in real life in such ways as: describing values of the variables on the web; describing information about web pages (content, author, created and modified date); present content for search engines and supporting electronic libraries. The key concepts of RDF are as follow (W3C 2004):

- Graph data model
- URI-based vocabulary
- Data types

3.1.1 Graph Data Model

The idea of RDF data model has become a general web resource description and modelling language. It can be used for conceptual description or the modelling of information stored in web sources. RDF is based on defining statements about resources in the form of subject-predicate-object expressions. Presented as Figure 3-1 below, the property (predicate) expresses the relationship between the subject and object. "Each triple represents a statement of a relationship between the things denoted by the nodes that it links." (W3C 2004) As doctor Sikos stated (Leslie, 2013), "Any expression in RDF is a collection of triples. A set of triples is called an RDF graph, which is a directed, labelled graph that represents information on the web."

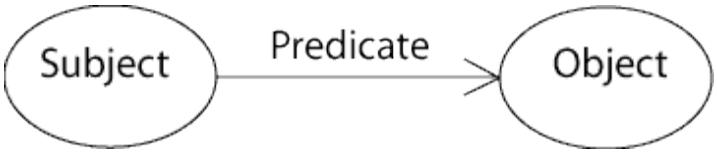


Figure 3.1 Subject-predicate-object expressions

3.1.2 URI-based Vocabulary and Node Identification

In brief, a triple is the smallest irreducible representation for binary relationship. However, what makes RDF triples special is that every part of the triple has a URI associated with it. In order to make this to seen more intuitional, we will introduce a very simple example given by Nico at www.stackoverflow.com website:

The statement "Mike Smith knows John Doe" might be represented in RDF as:

uri://people#MikeSmith12 http://xmlns.com/foaf/0.1/knows uri://people#JohnDoe45
(Adams, 2012)

Seen from the example displayed above, it has three triples: subject (Mike Smith), property (knows) and object (John Doe). Each triple has a URI associated with it, and the second triple presents the relationship between the first and the third triples.
A node may be a URI, a literal, or blank. URI references are properties; it is used as a node, which identifies what that node presents. A URI reference predicates and identifies a relationship between the things, which connected by the nodes. A predicate URI reference can also be a node in the RDF graph. According to Dr. Sikos (Leslie, 2013), RDF data model can be used for describing any kind of resources or information that can be identified by a URI.

3.1.3 Data types

In computer programming, a data type is a classification identifying one of various types of data. Similarly, RDF uses data types in presentation of values such as integers, floating point numbers and dates. In RDF, a data type consists of a lexical space, a value space and a lexical-to-value mapping. Built-in concept such as numbers dates or

other common values do not exist in RDF data type. RDF identifies data types with URI references. (W3C 2004)

3.2 What is SPARQL?

SPARQL is a query language that has been designed for managing data in databases. With SPARQL, we are able to retrive, and manipulate data stored in RDF format; express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. According to www.W3.org, "SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions (www.w3.org, 2013)." The results of SPARQL queries can be either results sets or RDF graphs.

In order to understand RDF and SPARQL in a more objective view, we will introduce a short example of SPARQL later in the text. Before we explain how SPARQL works, it's important to understand the following concept:

- Triple patterns
- Graph patterns = sets of triple patterns
- Queries Read from graphs

In general, most forms of SPARQL query contains a set of triple patterns called basic graph pattern. Similarly to RDF triples except that each of the subject, predicate and object may be a variable. (www.w3.org, 2013) For example we have data as follow:

Data:

```
<http://xyz.com/employee#000001> <http://xmlns.com/foaf/0.1/name> "Rajesh K Arora" .
```

SPARQL query:

```
select ?name where {
<http://xyz.com/employee#000001> <http://xmlns.com/foaf/0.1/name> ?name
}
```

Screenshot 3-1 SPARQL query get employee 000001

By using the SPARQL query bellow, we will find the name of the employee 00001 from the given data graph. The query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern. (Prud'hommeaux, Seaborne, & Laboratories, 2008)

The result:

name
"Rajesh K Arora"

Screenshot 3-2 Get employee 000001 result

However, the simple queries with simple match is not always sufficient to the real world task. The result of a query is a solution sequence, corresponding to the ways in which the query's graph pattern matches the data. There can be zero, one or multiple solutions to a query. Related to the example above, if this employee Rajesh K Arora belongs to a group, there will be another graph which links to the group. As a result, we will have this employee's name and group number while excuting a query through the multiple graphs with multiple matches. Multiple matches has been used in the IBM RDF application tutorial.

3.3 What is DB2?

DB2 is a series of relational database management products of IBM; it provides a multi-structured data server that allows for storing relational data as well as hierarchical data, natively. (Chong; Wang; Dang; & Snow, 2007) DB2 runs on multiple platforms including: Linux, UNIX or Windows, System z (DB2 for z/OS) and system i (DB2 for i5/OS). It provides the interface for programs to access and manage data through SQL and other query languages.

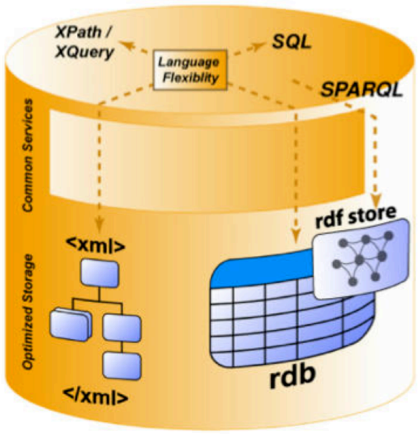


Figure 3.2 RDF on DB2

As shown in Figure 3-2 above, In DB2 version 10, it does not only support relational database, with a single platform, it has RDBMS, XML and RDF support. The XQuery language support manages XQuery data, SQL support manages the relational data, and SPARQL support manages the RDF data. In this study, we will focus on the RDF support in DB2.

DB2 also provides security for data and has utilities to maintain the data. Compares to Oracle, DB2 has a different data server structure. Oracle is using their unique system ID, which can be accessed by one and only one database at a time in a non-RAC (Real Application Clusters) environment. On contrast, DB2 instance can access more than one database at a time, even though most production database however is managed by one instance. To sum up what DB2 is, we can briefly translate it into two words: Integrated and Open.

Integrated: IBM DB2 provides highly integrated performance for data management. Through several features, for instance: DB2 software has built-in support for Microsoft.NET and Java development environment; DB2 is integrated into WebSphere, Tivoli, Lotus, and Relational products. It provides support for heterogeneous data sources for both structured and unstructured information. In addition, the DB2 family has cross-platform capabilities and can be integrated natively with web services and message-queuing technologies. (Chong, Wang, Dang, & Snow, 2007)

Open: DB2 software allows for different technologies to connect and integrate by following standards. It provides strong supports for Java and Linux operation system, XML, Web service, grid computing and other major industry applications. In our research, we have used DB2 Express C, which is a no-charge version of DB2 Express Edition that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

3.4 Other involved software

Jena and ARQ

In the tutorial, we have used JENA 2.6.3 package and ARQ 2.8.5 package while setting up the environment for DB2 RDF support. Jena is a Java framework for building Semantic Web applications, it provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and servers. Discussed earlier, SPARQL is the query language to retrieve and manipulate data stored in RDF format. And ARQ is a query engine for Jena, which supports the SPARQL RDF query language. The main feature of JENA and ARQ are listed below (Jena.apache.org 2012):

Jena

- An API (Application Programming Interface) for reading, processing and writing RDF data in XML, N-triple and turtle formats;
- An ontology API for handing OWL (Web Ontology Language)and RDFS (RDF schema) ontologies;
- A rule-based interface engine for reasoning with RDF and OWL data sources;
- Allow large numbers of RDF triples to be efficiently stored on disk;
- A query engine compliant with the latest SPARQL specifications
- Allow RDF data to be published to other applications using a variety of protocols

*N-Triple: according to W3C, N-Triples is a line-based, plain text format for representing the correct answers for parsing RDF/XML test cases as part of the RDF Core working group. (W3.org, 2001)
*OWL: it is a language which builts on top of RDF, for processing information on the web. (w3schools.com, unknown)

ARQ

- SPARQL/Update;
- Access and extension of the SPARQL algebra;
- Property functions for custom processing of sematic relationships;
- Aggregation, GROUP BY and assignment as SPARQL extensions;
- Support for federated query and extension to other storage systems;
- Client-support for remote access to any SPARQL endpoint (not used in the tutorial);

*SPARQL/Update: it is an update language for RDF graphs. According to w3.org: the update operations are performed on a collection of graphs in Graph Store. Operations are provided to update, create and remove RDF graphs in a Graph Store. (W3C, 2013)

Joseki

Joseki is a SPARQL server for Jena and it is an HTTP engine that supports the SPARQL protocol and the SPARQL RDF query language. The joseki servlet can be insalled in any servlet cotainer or web application server, with Joseki, a processor can either execute queries on a fixed, predefined dataset, or it can dynamically assemble the dataset from the query, which depends on the query.

Joseki configuration file is an RDF graph. The default name is "joseki-config.ttl", which you will meet later in the implementation, when we set up the Joseki on the system. The configuration file is often written in Turtle or N3 format, rather than RDF/XML. (Joseki, 2010) There are two http protocols that joseki supports:

SPARQL protocols and a protocol for SPARQL/Update. In the tutorial, Joseki has been used to access DB2 database server by adding a service to the Joseki-config.ttl file and including several modifications in other joseki files. Recently, Joseki has stopped its developement, and it is replaced by Apache Jena Fuseki. (Joseki.org, 2010). However, by following the tutorial, we will still use Joseki as our SPARQL server in this implementation.

4 Implementation, Problems and Solutions

This study is based on the first version of the material provided by IBM, Resource description framework application development in DB2 10 for Linux, UNIX and Windows, which was published on 24 May, 2012 (IBM, 2012). It was noticed later by the author that IBM has an updated version of the tutorial as of 23 January 2013, however, within the time allowed for this study, it was not possible for us to restart the implementation all over again with the updated version. According to Mr Sahoo, a senjor software engineer of IBM, who is also one of the main developer responsibe for RDF application development in DB2 and this tutorial: the most important change in the new version of the tutorial is that IBM has changed name of the sql script in the RDFDB2Tutorial.zip file to make it consistent with the tutorial. Since in the original version of the tutorial we have experienced some problems with the inconsistency of the jar files. This specific problem will be addressed during the verification of the tutorial.

Detailed instruction of implementation will be indicated in this paper only when the steps are missing from the IBM tutorial. Thus, in another word, duplicated writing of the original tutorial will not be included. Some of the instruction can be very basic, since we have targed the audience to be whoever is interested in learning this technology. Therefore, if you are one of the professional user in Linux or DB2, please skip some of the instructions that you have already known. The reader must keep in mind that, this paper is examing the first version of the tutorial, thus the results of this study refer to the first version of the tutorial only. In this chapter, we will start the implementation with the environment setup for the user to run RDF on DB2 environment.

Some problems have occurred during the implementation, corresponded solutions will be provided with the problems in this chapter. In order to verfiy the study result in later time, we will report what we have done on the system during the

implemenmtation, what has been done different to make the application work and what are the necessary steps that are missing from the tutorial.

4.1 Initial point

The purpose of this study is to examine the development of RDF on DB2 and it's process from installation to administration. The steps such as how to download a corresponding operation system (such as Windows or Linux) is not included in this paper. Thus the initial point for this paper is set to the environment set up for running DB2 on a virtual machine. The working computer is a Macbook pro laptop with Mac OSX version 10.8.3, Intel Core 2 Duo, 2.4 GHz processor speed.

4.2 Road Map

The tutorial has introduced a sample use case for an RDF application; a list of detailed activity on how to build and maintaining this application is as follow:

- Environment setup for running DB2 on your computer
- Creating an RDF store
- Inserting data into it
- Querying the data
- Updating the data
- Maintaining the store by ensuring that statistics required for good query performance are up to date.
- Converting the default store to Optimized store

The implementation of this RDF application development starts first with the environment set up for running DB2 on the local machine; some software has been downloaded as well as moved to the required place. After moving the files to the correct place, you need to set up the class-path, which tells where the Java virtual Machine or the compiler to look for the user–defined class and packages. The environment set up for executing DB2 RDF commands is completed at this stage. Once you set up the RDF environment on DB2, you can start running the bash files that are located inside the RDF folder.

According to the tutorial, you first need to run the bash file `createrdfstore` to create an RDF store called staffing for this RDF application in DB2. After you complete this step, you can start inserting data into the RDF store that you just created. With the employee data inserted into the store "staffing", you will be able to perform a series of operations on these data by executing SPARQL queries. In the tutorial, it has tested listing all the friends of the project members, which should return six members as a result. Furthermore, the tutorial has also instructions on how to insert and delete new triples into the existing graphs.

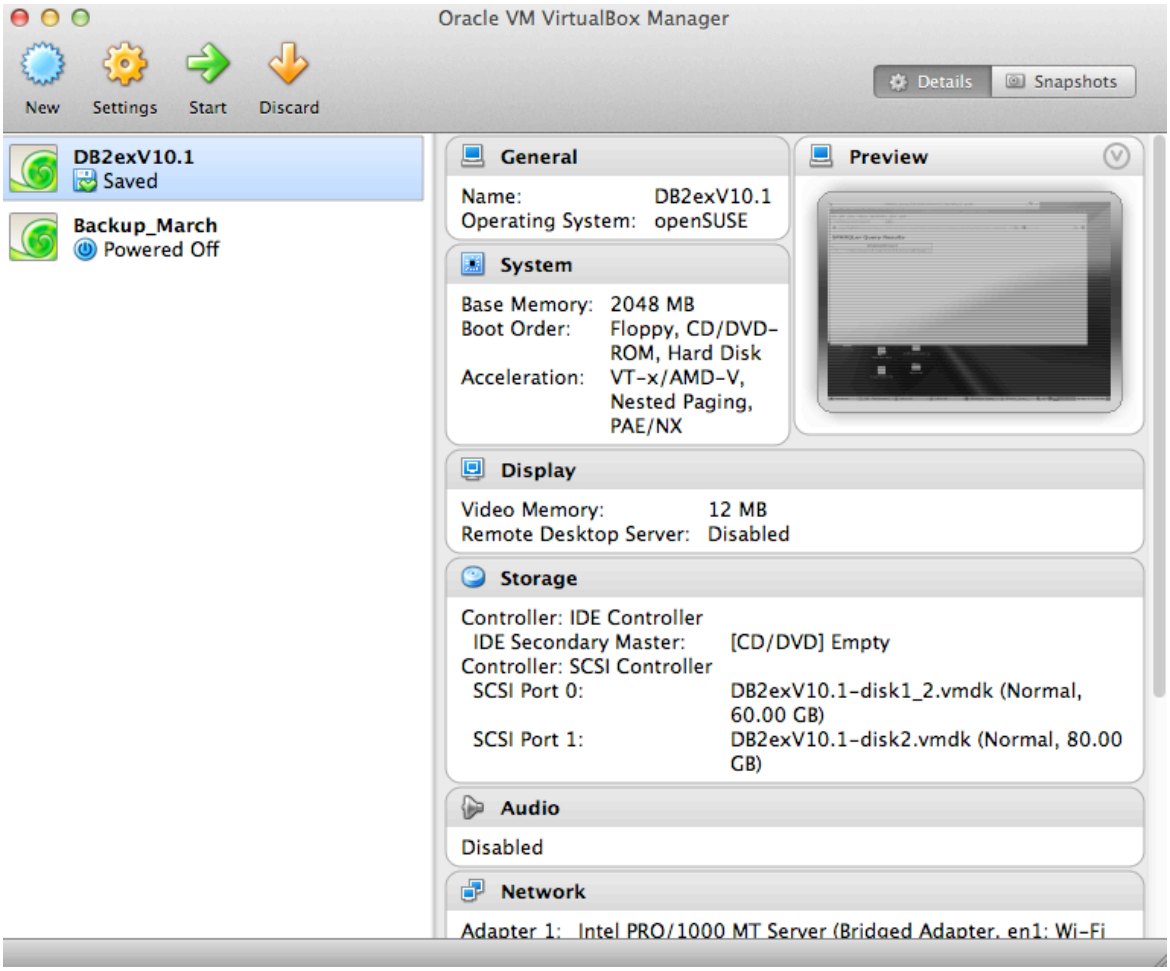
In the end, we will administrate the RDF store and convert the default store into optimized store. The db2rdf compiler for optimizing the query plans will use the statistics that are collected from DB2RDF. This helps in improving the performance of the SPARQL queries (Sahoo 2013).

4.3 Environment setup

In order to work with DB2, we have installed a clean version of OpenSUSE as our operation system through a virtual machine. In this study, we are using Oracle VirtualBox Version 4.2.6. The operating system is OpenSUSE with all the necessary software installed, such as DB2 10 Express-C and IBM data studio.

Setup virtual environment on Mac

Download the Oracle VirtualBox from the VirtualBox website and choose the right version according to your host computer. In this case, we are running on a Mac OS, so we have downloaded VirtualBox 4.2.6 for OS X hosts. After installing the virtual machine on your host computer, then you are ready to import the DB2 image to your virtual machine. Go to File → Import Appliance →Select the DB2 image and click import. A successful virtual environment setup should look like the image below:



Screenshot 4-1 Virtualbox setup

The virtual appliance that we are using in this study is DB2exV10.1 and we have one backup image of this project on the same virtual machine.

Setup RDF working environment on DB2

In this tutorial, a sample Java project is provided in order to interact between database and the RDF. You can download the DB2RDFTutorial.ZIP from IBM developer website, which contains this Java tutorial. The links to the downloads are provided at the end of the paper with other files. While performing the DB2 tasks in this study, we assume the use of the db2inst1 as the authorization ID, which has all required administrative privileges. To unzip the zip files in the same directory in OpenSUSE, use the following command:

```
db2inst1@db2v10:~> unzip DB2RDFTutorial.zip
```

The same command line applies to unzip the other required packages, such as JENA, ARQ, and Joseki if needed.

According to the tutorial, in order to set up the environment to use DB2 RDF support, you need to download the JENA 2.6.3 package, ARQ 2.8.5 package and commons-logging-1-0-3.jar file into the `sqllib/rdf/lib` folder. Use the following command to access as a root user:

```
db2inst1@db2v10:~> su
Password: <passsword>
... <working on root user privileges>
db2v10:/home/db2inst1 # exit
exit
```

sudo allows a permitted user to excute a command as the superuser or another user, as specified in the sudoers file. In this case, sudo allows you to access as the root user. As by being user db2inst1, it is not possible to move or copy files from one directory to another in this system, this is due to the permission setting on the computer.

Furthermore, since ARQ 2.8.5 is an older version, most of the download packages do not include the file arq-2.8.5.jar in ARQ 2.8.5 package. If it is so, please download the later version, for example ARQ-2.8.8 package instead. The rest of the files should be the same as what the tutorial displays except arq-2.8.8.jar instead of arq-2.8.5.jar.

As the last step of DB2 RDF environment set up is setting the correct classpath. To set classpath in Linux:

```
db2inst1@db2v10:~> CLASSPATH=$CLASSPATH:/home/db2inst1/sqllib/rdf/lib
db2inst1@db2v10:~> export CLASSPATH
```

which is different from the command line that provided in the tutorial for Windows operating system. Use `db2inst1@db2v10:~> echo $CLASSPATH` to verfiy whether your classpath has been correctly set up:

```
db2inst1@db2v10:~> CLASSPATH=$CLASSPATH:/home/db2inst1/sqllib/rdf/lib
db2inst1@db2v10:~> export CLASSPATH
db2inst1@db2v10:~> echo $CLASSPATH
/home/db2inst1/sqllib/java/db2java.zip:/home/db2inst1/sqllib/java/db2jcc.jar:
l/sqllib/java/db2jcc_license_cu.jar:.:/home/db2inst1/sqllib/rdf/lib
db2inst1@db2v10:~>
```

Screenshot 4-2 Classpath set up

4.4 Create an RDF store

While creating the RDF store as the tutorial indicated, we encountered several problems. The tutorial assumes that the DB2 server is on your local machine and is running at the default Windows port of 50000. However, we are running db2 on OpenSUSE and the port number is unknown. When you try to create the rdf store with the port number 50000, an error message appears as "Connection refused". Therefore, if you are running db2 on another operation system rather than Windows, or for some reason your createrdfstore command doesn't work, you should first check which is the port number that db2 runs on your machine. To find out which port number db2 runs:

Step1> `db2 "get dbm cfg" | grep SVCENAME`

Step2> `cat /etc/services | grep 'db2c_db2inst1'`

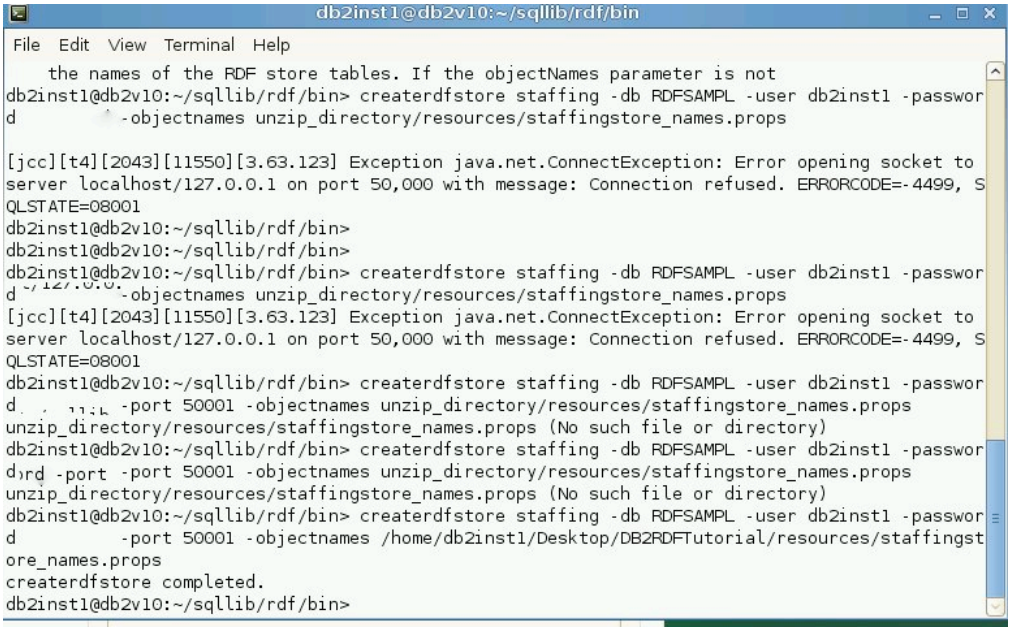
```
db2inst1@db2v10:~> db2 "get dbm cfg" |grep SVCENAME
TCP/IP Service name                          (SVCENAME) = db2c_db2inst1
SSL service name                         (SSL_SVCENAME) =
db2inst1@db2v10:~> cat /etc/services | grep 'db2c_db2inst1'
db2c_db2inst1     50001/tcp
db2inst1@db2v10:~>
```

Screenshot 4-3 DB2 server port number in Linux

Thus 50001 will be the port number for running DB2 server in Linux in this case and it is necessary to specify the port number while creating the rdf store. Use the following command:

```
createrdfstore staffing –db RDFSAMPL –user db2inst1 –password password –port
50001 –objectnames
home/db2inst1/Desktop/DB2RDFTutorial/resources/staffingstore_names.props
```

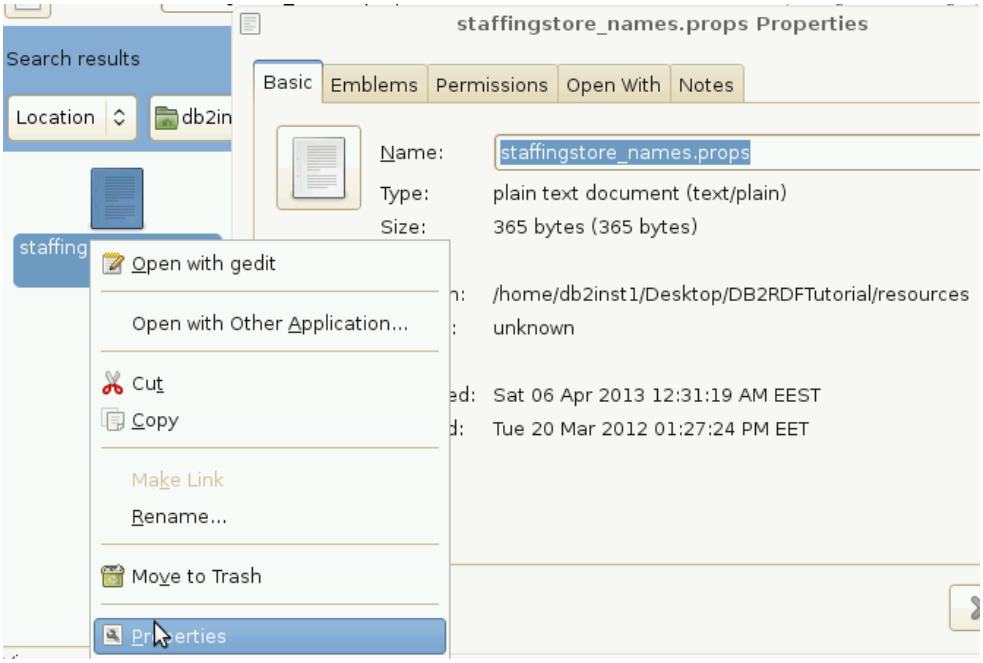
Screenshot 4-5 indicates the process in terminal:



Screenshot 4-4 createrdfstore

The file staffingstore_names.props contains the tables to be created and their table spaces. Seen from the screenshot 4-3 above, in Linux, we have used an absolute path of the file staffingstore_names.props when create rdf store "staffing". In our Linux experiment, the staffingstore_names.props is located in the resources folder under DB2RDFTutorial directory which we used as the unzip_directory.

In general, to find out where is your file located: search the file on your computer, right click to open properties of the file, copy paste the location path to the command line.

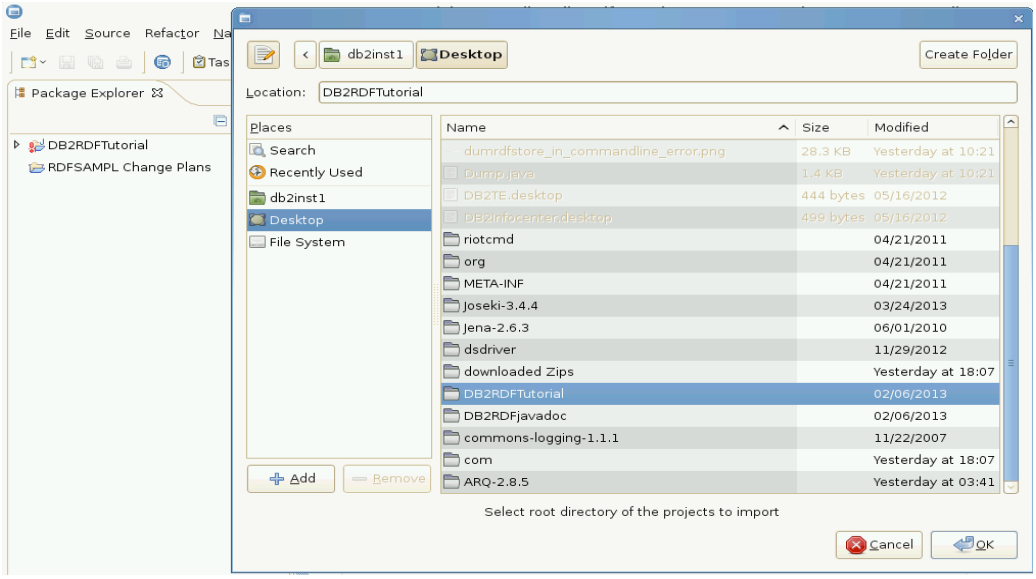


Screenshot 4-5 Find out correct path of the file

After fixing the port number of db2 and the location of the file, it became possible to run the -createrdfstore command and successfully created rdf store: staffing.

4.5 Loading data into the RDF application

If all the required jar files are added to the classpath correctly according to the tutorial, you can start to run the java project located in the DB2RDFTutorial package. Assume that you are already in the IBM DataStudio. Click File → Import → Existing Projects into Workspace →Select root directory. Browse to the file the DB2RDFTutorial and click ok.



Screenshot 4-6 Adding DB2RDFTutorial into IBM data studio

Imported java project will appear at the left side of the screen under the Package Explorer. The java programs are located under the sample folder which is under the src folder of DB2RDFTutorial. All necessary java programs are included with different functions, such as: deleting graphs and triples, insertting graphs and triples and query excectuor. Modify the java code according to your username, passport and port number to all the java programs that are provided by the DB2RDFTutorial:

QueryExcecutor.java | InsertGraph2.java

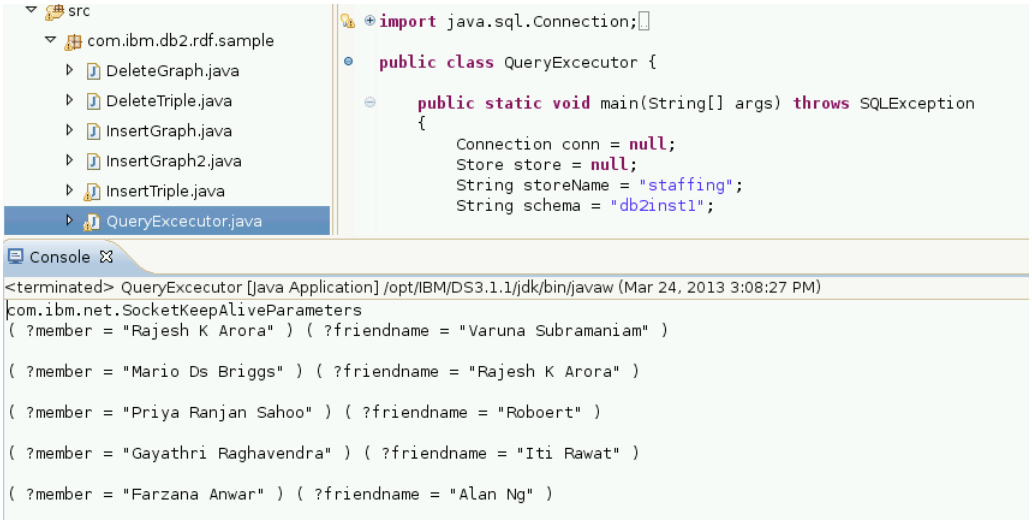
```
* Licensed Materials - Property of IBM
package com.ibm.db2.rdf.sample;
import java.io.IOException;
public class InsertGraph2 {
public static void main(String[] args) throws SQLException {
Connection conn = null;
Store store = null;
String storeName = "staffing";
String schema = "db2inst1";

try {
Class.forName("com.ibm.db2.jcc.DB2Driver");
conn = DriverManager.getConnection(
"jdbc:db2://localhost:50001/RDFSAMPL", "db2inst1", "password");
conn.setAutoCommit(false);
} catch (ClassNotFoundException e1) {
e1.printStackTrace();
}
```

Screenshot 4-7 insertGraph2.java

Please note not to run the InsertGraph.java (which is InsertGraph2.java in the picture) more than one time. Because the method: `staffingDefaultModel.add(merged);` adds all the merged in-memory model to the staffing model everytime when you excute it.

Otherwise it would bring duplicated data into the dataset, which will lead to wrong results.

The next step is to run the QueryExcecutor.java file with the correct username, password and port number. The result should be as follow:



Screenshot 4-8 QueryExecutor.java

4.6 Executing SPARQL queries over HTTP

4.6.1 Set JOSEKIROOT

In the tutorial, we are using Joseki HTTP engine to execute the SPARQL queries over HTTP against the DB2 database. Joseki provides a web interface for performing SPARQL queries on an RDF graph. In order to set up Joseki for this task, the tutorial has listed the necessary steps. In step 3. Set the JOSEKIROOT variable to the current folder, it did not give detailed information of how to set the JOSEKIROOT. Here is an explaination on how to set it up:

```
export JOSEKIROOT=/home/db2inst1/Desktop/Joseki-3.4.4
export PATH=$PATH:$JOSEKIROOT
```

Use your own location path for Joseki package instead of the path above. To find out the full path of the Joseki package, right click the package, choose properties and the location is listed as "Location". For the purpose of rapid phototyping, a shell script can

be created here to set up Josekiroot, instead of typing the command like above everytime connect to the rdf server.

By setting up the JOSEKIROOT in the system successfully, you will be able to run the RDF server. To be notice that, in order to connect to the RDF server everytime, one should always set the JOSEKIROOT before connecting to the RDF server as follows:

```
cd $JOSEKIROOT
bin/rdfserver
```

To disconnect the server, use `Ctrl+C`. While the rdf server is connected, the Joseki main page should appear through the URL: localhost:2020.

4.6.2 Adding a service for DB2 in Joseki

In order to access the DB2 database server, you need to modify Joseki. The purpose of adding a service for DB2 in Joseki is adding a SPARQL processor for DB2 RDF store. It requires store name, username, password and schema in which the store exists.

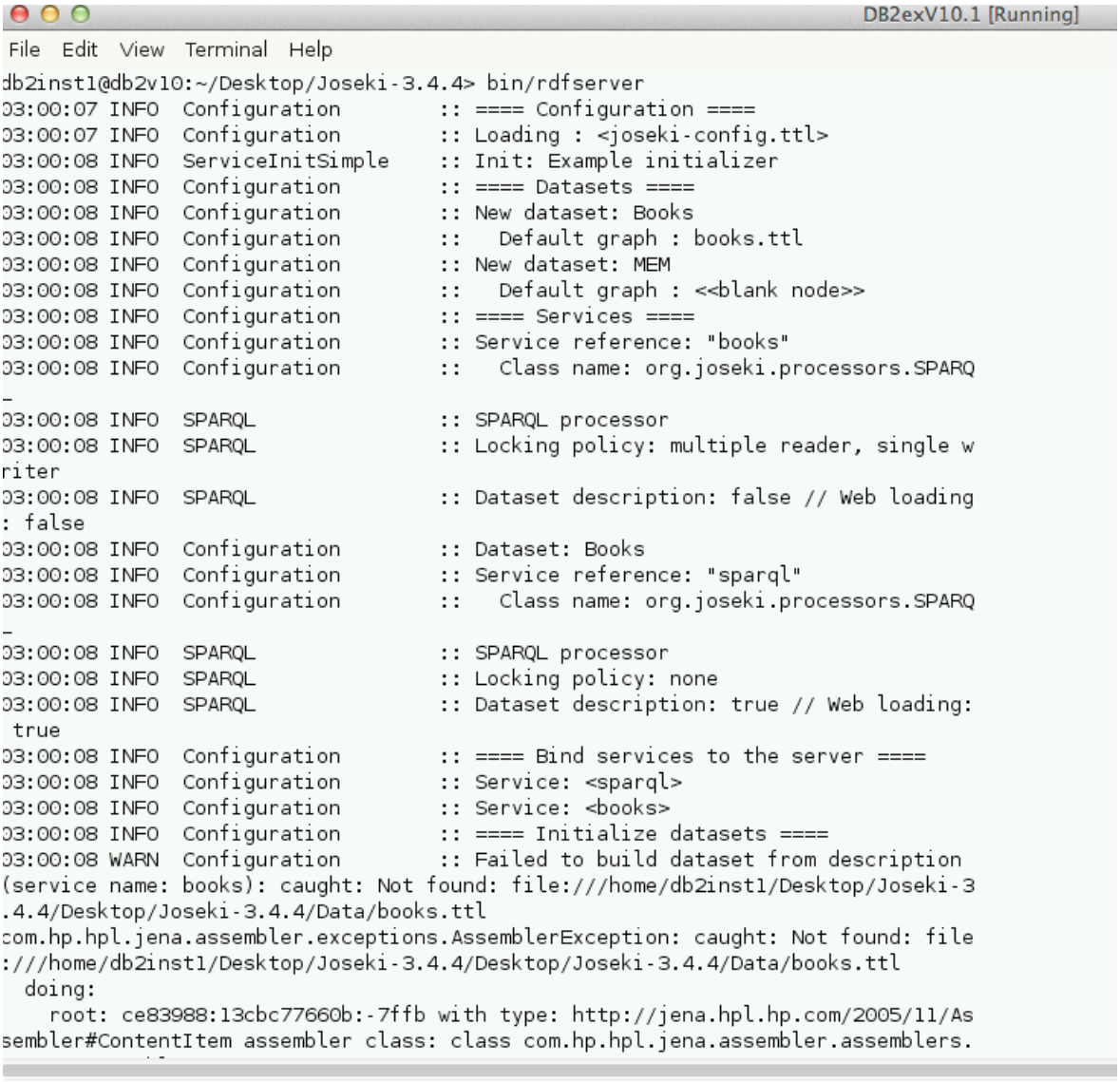
Below there is a copy of the code from the joseki-config.ttl file that we have used for adding service for DB2. You can copy this part and add it to your joseki-config.ttl file. Beware of the dot (.) used as the terminator character in the code, if you are not familiar with the SPARQL query language. Change the username, password, schema and port number according to the information on your own system.

```
#service for DB2 - SPARQL processor for a DB2 RDF store @prefix db2rdf:
<http://rdfstore.ibm.com/IM/joseki/configuration#> .
<#serviceDB2>
rdf:type      joseki:Service ;
rdfs:label    "SPARQL against DB2 RDF store" ;
joseki:serviceRef "db2";

# web.xml must route this name to Joseki
joseki:processor          joseki:ProcessorDB2SPARQL ;.
joseki:ProcessorDB2SPARQL
rdfs:label    "DB2 RDF General SPARQL processor" ;
rdf:type      joseki:Processor ;
module:implementation                joseki:DB2ImplSPARQL ;
db2rdf:jdbcConnectString             "jdbc:db2://localhost:50001/RDFSAMPL";
db2rdf:userName "db2inst1" ;
db2rdf:password "password" ;

#db2rdf:jndiDataSource               "testDS" ;
```

```
#RDF data set details: storeName and schema in which the store exists
db2rdf:storeName "staffing" ;
#db2rdf:schema "db2inst1" ;.

joseki:DB2ImplSPARQL
rdf:type      joseki:ServiceImpl ;
module:className
<java:com.ibm.rdf.store.jena.joseki.SPARQLProcessor>.
```

According to the tutorial, on page 17, step 7, several jar files are required to be moved to the Joseki folder, includes the db2jcc4.jar file which we have set the classpath to earlier in the tutorial. Do not move these jar files from the original place to the Joseki folder, instead, copy the jar files and add them to the required folder. Because, if you move the jar file from the original place, especially moving the DB2 JCC driver file, it will cause a problem when binding the files later in the tutorial. A connection error will appear in the system.

A successful set up on Joseki should bring you to the sample output when you restart the Joseki. After following the step 9 to 11, an error has occurred while binding the services to the server. As you can see from the following screenshot, that service for DB2 has not been binded:



Screenshot 4-9 Joseki server setup

It was not described in the tutorial, but if you encounter a similar problem, please go to the Joseki folder and check in the joseki-config.ttl file. I have found that the location path of the file books.ttl is different from where it actual located. So I have rewrote the path location in the joseki-config.ttl file according to its correct path.

```
## ------------------------------------------------------------
## Datasets

<#books>   rdf:type ja:RDFDataset ;
    rdfs:label "Books" ;
    ja:defaultGraph
      [ rdfs:label "books.ttl" ;
        a ja:MemoryModel ;
        ja:content [ja:externalContent <file:Data/books.ttl> ] ;
        #ja:content [ja:externalContent <file:Desktop/Joseki-3.4.4/Data/books.ttl> ] ;
      ] ;
    .

<#mem>  rdf:type ja:RDFDataset ;
    rdfs:label "MEM" ;
    ja:defaultGraph [ a ja:MemoryModel ] ;
    .

## ------------------------------------------------------------
```

Screenshot 4-10 Changes in joseki-config.ttl

Beginners tip:

To open and edit a file: `- gedit filename`

Copy files to another directory: `sudo cp filename destination_directory`

4.7 Administering RDF stores

Stated by Mr Sahoo, a senior software engineer from IBM, who is one of the author of the tutorial: The statistics that are collected in DB2RDF are used by the db2rdf compiler for optimizing the query plans. This helps in improving the performance of the SPARQL queries.

These statistics are stored in the following two tables in the db2 database:

`<StoreName>_basestats`
`<StoreName>_topkstats`

Basestats stores the average statistics like (count of triples per subject , object or predicate etc) and Topkstats stores the frequently occurring subject , object or predicate and the count of their occurences.

According to the tutorial, DB2 RDF store uses two sets of distribution statistics for improving performance of the SPARQL queries: DB2 table statistics and RDF store statistics. (IBM 2012) The DB2 table statistics is done automatically during store

creation, to enable such automated statistics collection, the user has to set up the automated statistics collection on the system. The second option is the RDF store statistics, which is done manually. The tutorial has a mere brief instruction of how to set up the automated statistics collection. For the ones who are not familiar with DB2 or Linux, we will provide a detailed explaination on how to set up the automated statistics collection and manual statistics collection.

To set up automated statistics collection on DB2 database server, the following pre-work has to be done:

1. Turn on the Administrative Task Scheduler.
   The administrative task scheduler enables DB2 database servers to automate the execution of tasks. It allows you to build applications that can take advantage of the administrative task scheduler through a programmable SQL interface. (IBM unknow) To turn on the administrative task scheduler, issue the following command: `ad2set DB2_ATS_ENABLE=YES`
2. Activate the database.
   To activate a certain database, the user first needs to activate the db2 command line processor (CLP), which represents an interface to access DB2 UDB functions. (IBM 2005) To start the command line processor use command: `db2`, issue the command line: `ACTIVATE DATABASE RDFSAMPL` to activate the database called RDFSAMPL, which is the sample database in the tutorial. To deactivate the database, issue the command line: `DEACTIVATE DATABASE RDFSAMPL`. (IBM 2008)
3. Ensure the privileges of the user
   Users can successfully execute operations only if they have the authority to perform the specified function (IBM 2008). To ensure the user (in this case is db2inst1) has update privileges on table SYSTOOLS.ADMINTASKSTATUS use the following GRANT command
   ```
   GRANT UPDATE ON SYSTOOLS.ADMINTASKSTATUS TO USER db2inst1
   ```
4. Schedulling statistics collection
   Automatic statistics collection is a background process which runs periodically. (IBM 2007) Setting up a schedule statistics collection on the RDF store we

created, enables the possibility of gathering statistics of the store every amount of time every hour. For example, the tutorial has set the automated statistics collection to gather statistics in every 15 minutes per hour.

To set up the schedulling statistics collection, issue the following command under `rdf/bin` directory with port number specified:

```
setStatsSchedule staffing -db RDFSAMPL -user db2inst1 -password password -port 50001 -schedule "15 * * * *"
```

Successfully scheduling statistics collection set up should bring you the following statement as in the screenshot below: SetStatsSchedule completeted.

```
db2inst1@db2v10:/opt/ibm/db2/V10.1/rdf/bin> setStatsSchedule staffing -db RDFSAM
PL -user db2inst1 -password password -port 50001 -schedule "15 * * * *"
SetStatsSchedule completed.
db2inst1@db2v10:/opt/ibm/db2/V10.1/rdf/bin>
```

Screenshot 4-11 SetStatsSchedules completed

5. To set up manual statistics collection gathering statistics for the RDF store, follow the instruction on the tutorial, but specified the port number as the way you run all the other bash file in the rdf folder. A successful manual statistics collection set up should return you the following picture:

```
db2inst1@db2v10:/opt/ibm/db2/V10.1/rdf/bin> ls
createrdfstore           loadScripts         reorgrdfstore        setUp
createrdfstoreandloader  registerrdfudf      reorgswitchrdfstore  updaterdfstorestats
droprdfstore             reorgcheckrdfstore  setStatsSchedule
db2inst1@db2v10:/opt/ibm/db2/V10.1/rdf/bin> updaterdfstorestats staffing -db RDFSAMPL
-user db2inst1 -password password -port 50001
updaterdfstorestats completed.
db2inst1@db2v10:/opt/ibm/db2/V10.1/rdf/bin>
```

Screenshot 4-12 updaterdfstorestats completed

4.8 Migrating to the DB2 database server from other RDF storage engines

According to the tutorial, it is possible to import your RDF data from another storage engine to DB2 database server. You can follow the tutorial from Populating a new RDF store if you already have a dataset exported from somewhere else. However, the tutorial does not provide detailed information on how to export the dataset on DB2. There are at least two possible solutions based on the theory:

1. Creating a java file that exports the dataset from DB2 into an N-triple file;

2. Create a bash file in rdf folder with the command line: 'com.ibm.rdf.store.cmd.DumpRdfStore' or run 'com.ibm.rdf.store.cmd.DumpRdfStore' directly from the command line.

Solution one:
Create a java program that dump out the data from DB2. The source code of this java program that we have used in this study can not be pasted here since it is one of the IBM unpublished file. However, you can create your own java program for getting the data from DB2. I have made a copy of the dataset when loading the data into the RDF application. To verify the data was correctly added to the DB2 database, we have dumped the data to the concole once. Select all the triples of the dataset, copy and paste the dataset into a notepad, in the end save the file to an N-quad file as exported.nq.

Solution two:
If you open any of the shell scripts in the rdf/bin folder, u will see that they invoke the 'setUp' script in the same folder. Open the setUp script and notice that it appends all the jars in the 'rdf/lib' folder to the classpath. Thus the final classpath setting is all jars from rdf/lib plus whatever is the existing classpath. And this is the classpath required to run any of the processes. It is possible to make a bash file just like the other bash files in the rdf folder, set the permission according to the other bash files and run the script.
The bash file that I have created for dumprdf is as follow:

```
#!/bin/sh
RDF_BIN=`dirname "$0"`
source "$RDF_BIN"\/setUp
java com.ibm.rdf.store.cmd.DumpRdfStore /home/db2inst1/Desktop/export.nq $*
```

Replace the absolute path with your own, depends on where do you want to export the N-quad file.

5 Verification of the experiment

Throughout in this chapter we will evaluate what we have done on the system during the experiment and verify the result of the RDF application development that we performed on DB2. The experiement of the implementation is based on the IBM tutorial, RDF application development in DB2 10 for Linux, Unix and Windows Part 1 and Part 2. In order to achieve a thorough evaluation of the result, we have implemented three variables structure during the experimentation. These three variables are independent variables, controlled variables and dependent variables. (Explorable.com 2011) The independent variable is what you change in order to provide a result. In this case is the data that we insterted into the RDF store. Controlled variables are the things that never change. In the implementation, we have decided to perform this task in DB2 environemnt instead of other database management environment. And we have taken Linux operation system as our operating system rather than Windows and other operating systems. So our controlled variables are DB2 and Linux SUSE operating system. The last but not the least, the dependent variable is what we are measuring. In this study, we aim to prove that the RDF application works on DB2 as the tutorial described, at the same time, we measure the results while excuting different queries. This shall also help us on understanding the structure of the N-triple data that has been inserted earlier to the RDF store staffing. The result will be narrowed down to four sections: Loading data into RDF store, Executing SPARQL queries over HTTP, RDF store administration and improvement.

5.1 Loading data into RDF store

During the implementation of RDF application on DB2 Linux for the past months, we have acknowledged a large amount of theoritical and practical knowledge in RDF, DB2 as well as SPARQL query language. The first result of the experiement is successfully loading data into the RDF store in DB2.

In this section, we have used a java program called InsertGraph2.java to insert the initial data from the four system into the RDF store. The program first created a

connection between DB2 database server and RDF programs, thus it allows the java program to connect to the store. The connection has to be created everytime when someone wants to work on the store. The program also creates an in-memory model for each N-triple file. And in this tutorial, we used JENARiotLoader .read method to read fro the N-triple files. The program merges four models into a single model, at the meanwhile, it adds this merged model to the default model in the staffing store. We have verified the data is stored in DB2 by dumping out the data from DB2. The result showed that all the data are added into DB2, however, the order of the graphs is placed differently from what the IBM tutorial received. We have manually gone through the data out put from the console to make sure that the elements we received are the same compare to what the tutorial presented. At this stage, we verified that the triples from all the four N-triple files are loaded into the default graph in the staffing system.

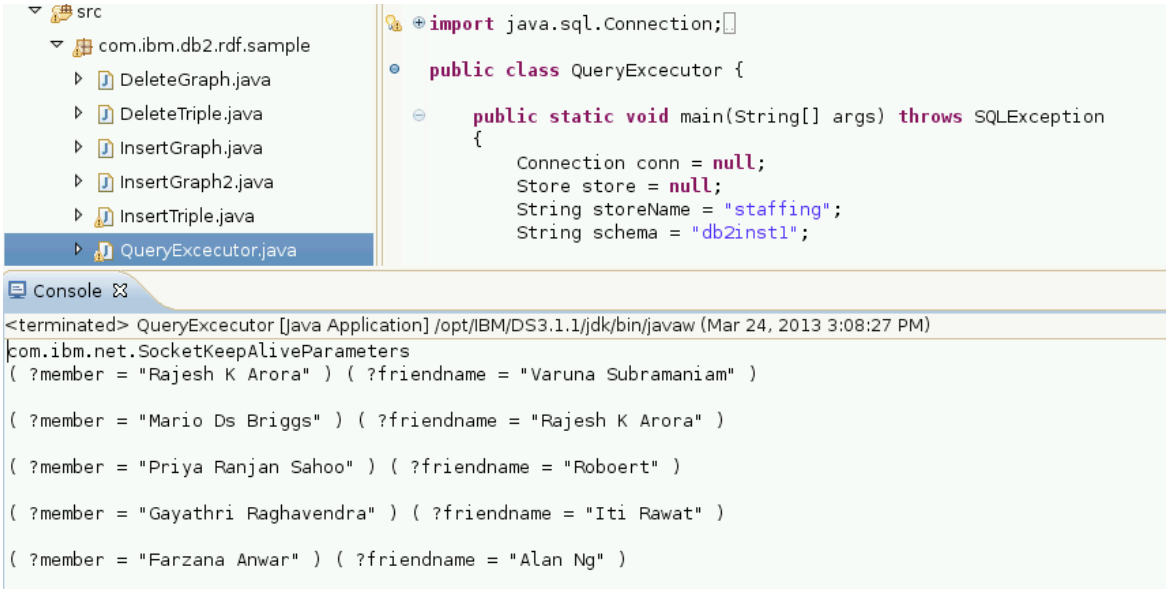
The next stage is to use SPARQL queries in the staffing system application. The tutorial has given one example query to find all the friends of the project members. We will use this query as our first test on the RDF application that we have built and compare the result with what the tutorial has presented. As mentioned earlier in this chapter, in order to reach a more profound and valid result on the experiement, it is necessary to have different independent variables while verifying the result. With changed variables, we will be able to receive a comprehensive interpretation of the results.

Please read through the tutorial, if you are not fimilar with the fictitious example that we are using in the tutorial as well as in this study. The link to the tutorial is listed at the end of this paper and the description of the fictitious example is placed in section 2 of the tutorial: The example application development use case. Since the members of the project RobotX are spread across the existing HR and Projects systems. In the past, it would have required an application programming interface (API) or other kind of integration between the system to get the resulting friends. Be that as it may, with the new staffing system, we have only used a simple SPARQL query below to fetch this information:

```
select ?member ?friendname where {
```

```
<http://xyz.com/project/robotX> <http://xyz.com/project/member> ?memberId .
?memberId <http://xmlns.com/foaf/0.1/name>  ?member .
?memberId <http://xmlns.com/foaf/0.1/knows>  ?friendsId.
?friendId      <http://xmlns.com/foaf/0.1/name>  ?friendname .
}
```

Run the above SPARQL query contained java program you will receive the following result (above is only part of the code where the SPARQL query is placed, the complete code of this java program is attached in the appendix):



Screenshot 5-1 QueryExecutor.java

The operation above proves the following three things:

1. The data has been added to the sample RDF store correctly
2. SPARQL query is working
3. By executing the SPARQL query, we received the expected result

Speaking of earlier statements using different independent variables, in addition to the SPARQL query that we just used in the test one above, we will test different SPARQL queries at the next verification, which is executing SPARQL query over HTTP. Because in order to prove the queries are working while executing over HTTP, the query has to work in the database itself first. Thus the duplicated verification on the database seperately is not necessary.

5.2 Executing SPARQL queries over HTTP

Our aim in this section is to exam the capability of executing SPARQL queries over HTTP on DB2. We plan to test this function with three different SPARQL queries, in order to prove that the function is not only working occasionally. In the last test, we have executed a SPARQL query to display all the friends of the project members, and last test proved that the query was correct as well as the dataset. Hence, in this test, we will first exam this same query over HTTP.

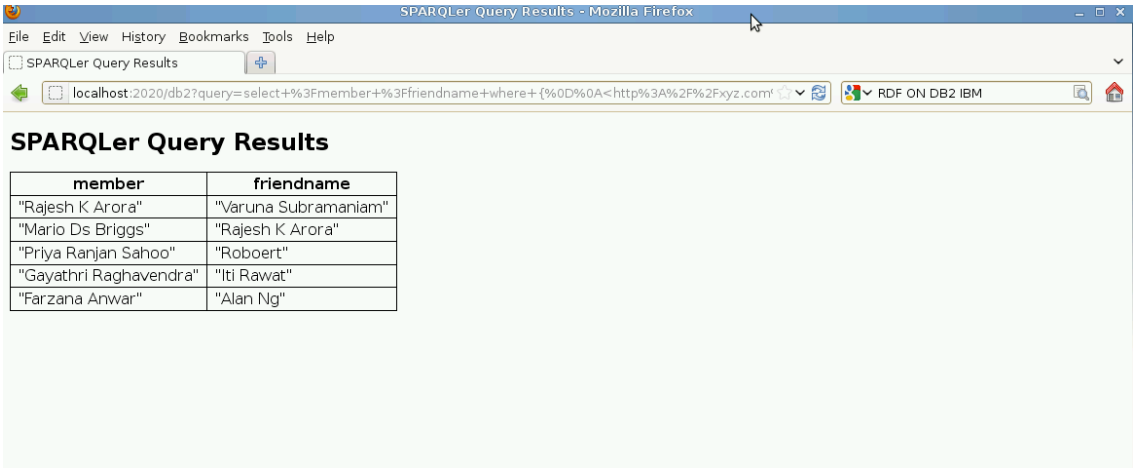
Based on the tutorial, we have used Joseki HTTP engine to execute SPARQL queries over HTTP against the DB2 database. In order to execute SPARQL queries over the web, you first need to set up its server Joseki. Mentiond earlier, Joseki is a SPARQL server for Jena and its server is configured with services. In our case, the Joseki server that we set up is implemented by a processor that execute queries on the dataset that we inserted into the store RDFSAMPL. The configuration file is an RDF graph named "joseki-config.ttl". In order to fulfill the action of executing SPARQL queries over HTTP, we first need to configure the Joseki server by adding a service for DB2 to the default configuration file "joseki-config.ttl", this provides a SPARQL processor for DB2 RDF store and DB2 processor support using FROM/FROM NAMED in the request. Second, in order to match the context path of url for maping servlets, an servlet-mapping entry for DB2 is required in the servlet-mapping area. It maps url patterns to servlets. For instance, when we send a request by input a SPARQL query into the web, servlet container decides to which application it should forward to. Last but not the least, change the action attribute of the form element from SPARQL to DB2 database server, so the server knows where to send the form-data when a form is submitted. The actual steps of how to set up Joseki has been detailed in the Chapter 4, 4.6 Executing SPARQL queries over HTTP have detailed instruction on how to set up Joseki server.

We assum that, at this stage, your Joseki server is correctly set up with all required configurations. Open a broswer on your computer and go to this address:

http://localhost:2020/sparql.html. Insert the same sorting friends query that has been used in chapter 5.1 loading data into RDF store to the follwoing text box:

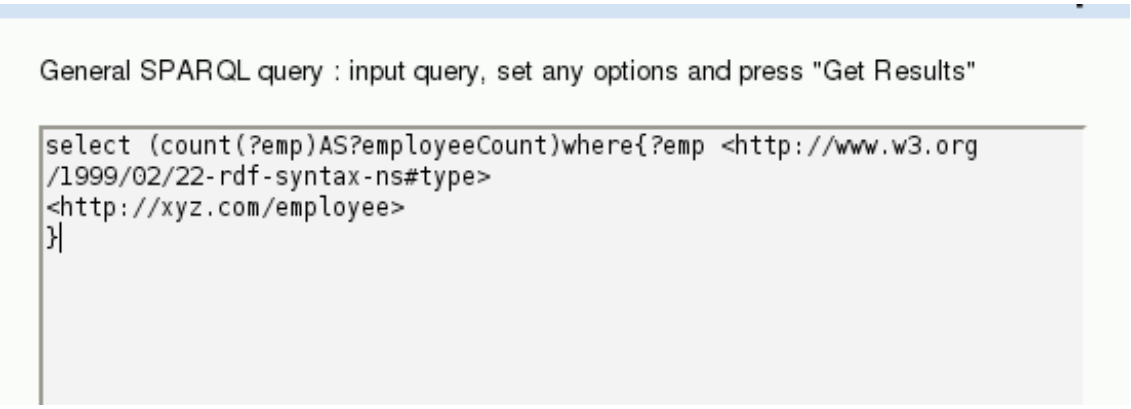
Screenshot 5-2 Finding friends query

By clicking the Get Result buttom at the lower part of the page, we will receive the following result on the page:

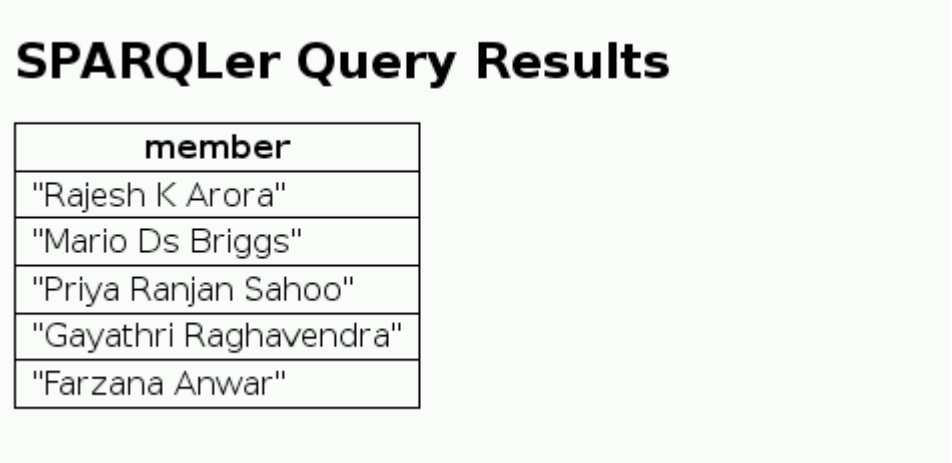


Screenshot 5-3 Finding friends query result over http

As planned, we have tested the same function with two other queries. The second query is as follow, and it aims to find out all the members of the project RobotX. This query was given by the IBM tutorial, on tutorial page 14. Input this query to the textbox as the Screenshot 5-4 shows. Click "Get Results" to display the result of the execution. Displayed by screenshot 5-5 Result of members of RobotX, the result contains five names, which are the members who participate in the project RobotX.

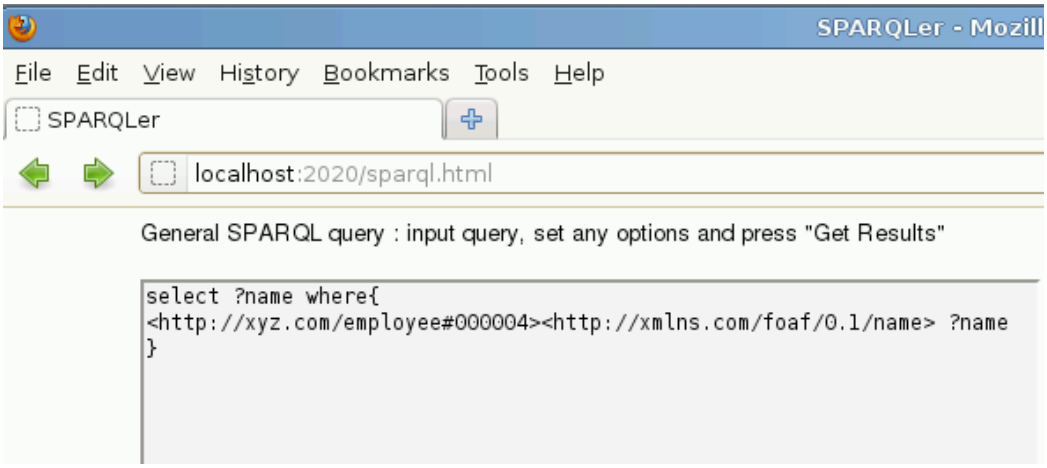


Screenshot 5-4 List of members of project RobotX

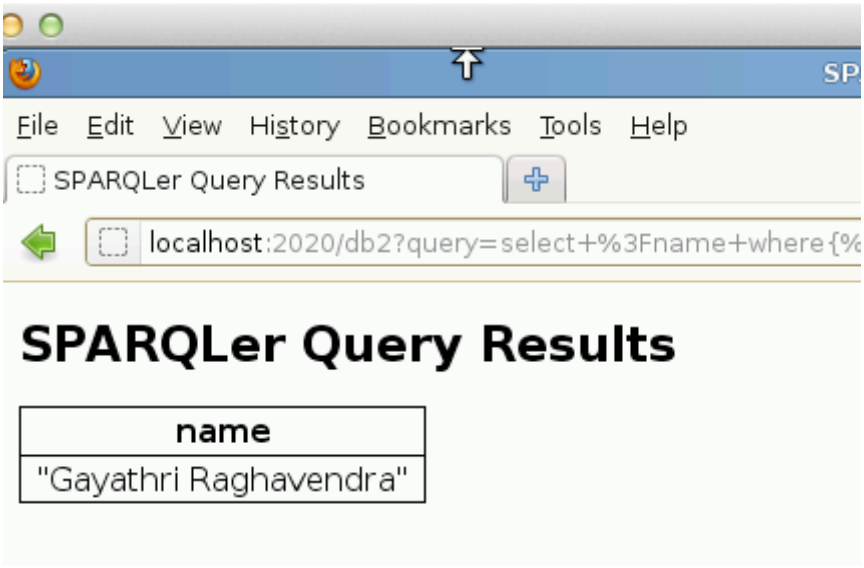


Screenshot 5-5 Result of members of RobotX

The last testing query aims to get the name of the employee who's employeeId is #000004. It is similar to the example that we have used when introducing SPARQL in the Chapter 3. By executing the query, we will fetch the name of the employee 000004 from the HR system. The query can be seen in screenshot 5-6: employee000004 query and the result is recorded in the screenshot 5-7: result of the employee000004.



Screenshot 5-6 employee000004 query



Screenshot 5-7 employee000004 result

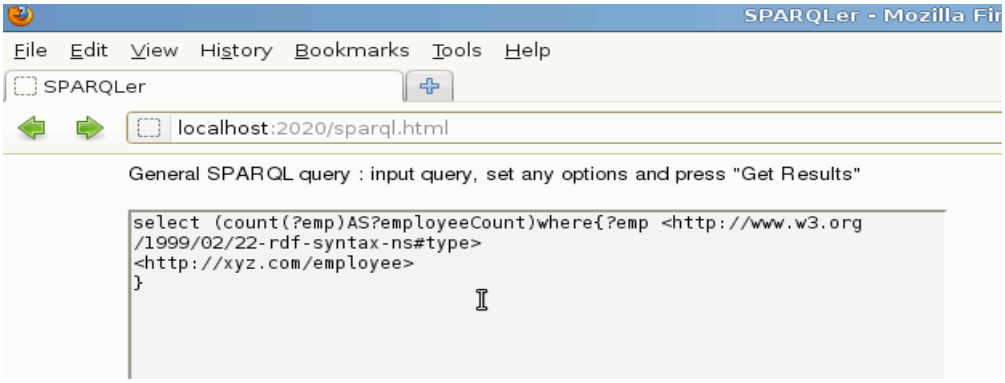
5.3 Updating data in the Staffing system application

Updating data includes inserting new triples into the existing graphs as well as deleting the triples from the existing graphs. We will start the second verification with inserting triples. The story behind is that the orgnization wants to add two new employees. According to the tutorial, the information of new employees are stored in an N-triples formated file called newdatat.nt, which is located in the resources folder of the tutorial. Adding these two new employee successfully to the origranization means that the information of this N-tripled file has to be added into the HR system. You can run the sample java project InsertTriple.java to accomplosh this process. Get the sample java project InsertTriple.java from the DB2RDFTutorial package. Remember, instead of

the relative path of the file newdata.nt, include the absoutle path of the file in this method:

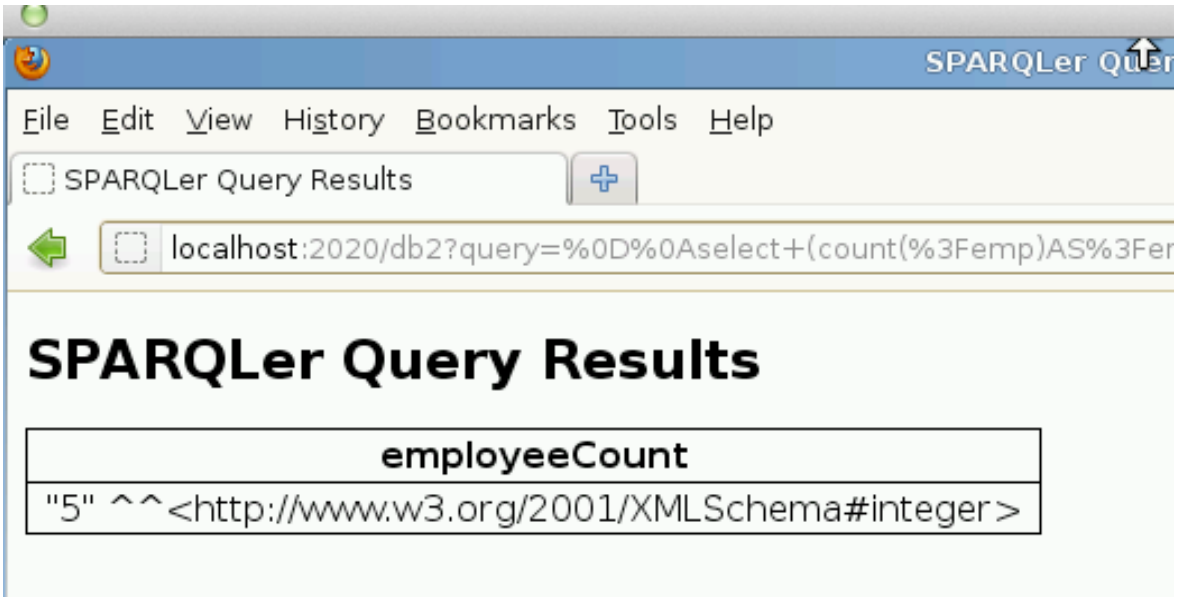
```
StmtIterator it = get-
NewTrplesToAdd("/home/db2inst1/Desktop/RDFDB2Tutorial/resources/newdata.nt");
```

With this method, the system reads information from the newdata.nt and return the list of triples which is included in this file. Related to the previous java programs that we have used in this study, in order to interact with the dataset, it is required to connect to the default model in the store. When the store is connected, the new triples then can be added into the store. To verify that the new data is inserted, count the employees in the staffing store before and after running the InsertTriple.java program, next compare the result. Use the following query to get the employee count:



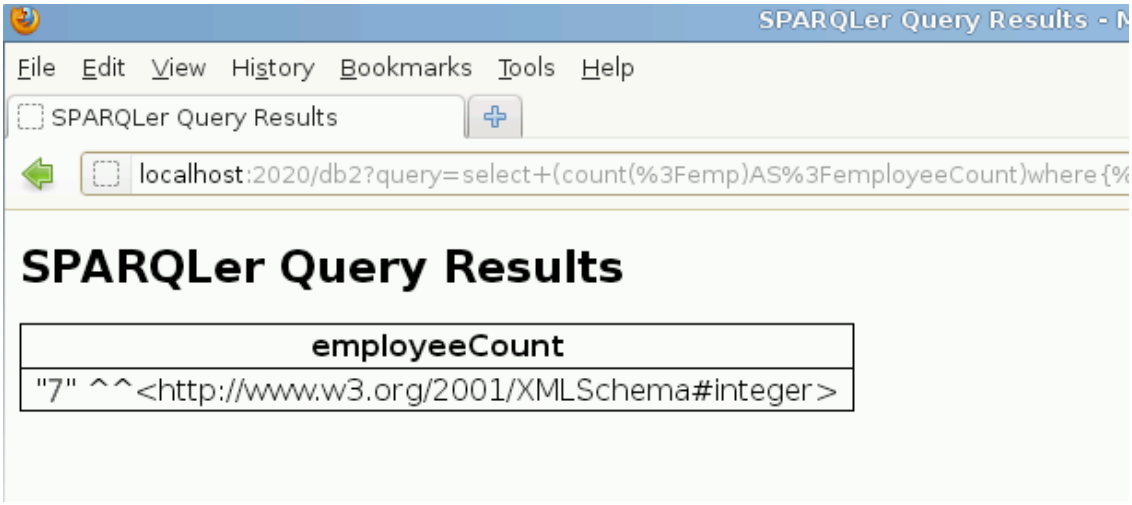
Screenshot 5-8 Employee count query

By clicing button "Get Result", we received the window below:



Screenshot 5-9 employee count old

Now we run the java program, InsertTriple.java in IBM data studio. Make sure that for all insert purpose java program in the tutorial, you should run it only once if succeeded. Because the repeated data will be added to the rdf store staffing when every next time you run the program. Assume that your InsertTriple.java program was run successfully, now execute the same employee count SPARQL query over HTTP. The result is shown as screenshot 5-9 below. It is noticed that the number of the employees has increased from five to seven after running the InsertTriple.java program. This proves that, the InsertTriple.java program works as it indicated and it is allowed to insert triples into an rdf store in DB2.



Screenshot 5-10 employee count new

The next step is to verify the deletion of the triples in RDF. Since we have just added two new employees to the store staffing in the last verification. Now we can test the deletion function by deleting the two new added employees from the rdf store. Similarly, run the DeleteTriples.java file that came with the RDFDB2Tutorial folder. The java file will first connect to the model, it will then look for the employee Id that was specified in the deletion program and in the end, this particular employee will be delete from the store. We have checked content of newdata.nt, as you can see from the screenshot 5-11: new employees' information, the new added employeeId are 000006 and 000007. Replace the triple in the DeleteTriples.java file by the new employees' triple. Run this java program seperately with employee Id 000006 and 000007. To verify the deletion of the triples, count the employees by executing the same employee count query that has been introduced earlier. The result of the employees count returned us a 5, Thus this can prove that the triple deletion is also working for RDF in

DB2.

Screenshot 5-11 new employees' information

We did not test the Deleting graphs method as the tutorial indicated of RDF in DB2 due to the limited time of the research. The current state of the rdf store needed to be saved, because a teacher from the university will use the virtual image of this implementation during his lecturing in June, 2013. In order to avoid any extra hassles within the time allowed, we did not test this function.

However, instead of examining the function of Deleting graphs in DB2, we have tested this same function by running another method at the early stage of the implementation, and it achieved the same result, which is removing all the triples from the graph/store. According to W3C (Patrick , 2004), an RDF graph is a set of triples, and an RDF store is a database where it stores and manages the data (triples). Thus, in the tutorial case, deleting the rdf store staffing from DB2 realizes the same result of deleting graph.

The initial reason of removing the rdf store staffing from DB2 was because one mistake that we made during the implementation. While loading the data into the rdf store in the section three of the tutorial, we received a dataset result which had differences on the sequences of the triples (the output includes all the triples though, but sequences is different). We were hoping to receive the exact same result (data and sequences) as displayed on the tutorial, so we ran InsertGraph2.java for too many times. However, aforementioned java file, InsertGraph2.java, adds the merged dataset into the rdf store staffing when every time you runs the program. As a result, we received a real large volume of the dataset with the same graph multiplied several times. This large dataset leaded us to a large repeated result on the QueryExecutor.java as well. It would have

resulted in an unnecessary use of time by deleting the triples or graphs one at a time. Therefore, we have deleted the whole rdf store staffing by running a single method:

```
// Delete store if required.
StoreManager.deleteStore(conn, schema, storeName);
```

After deleting the rdf store, run the store creation again and load all the data into the store through InsertGraph2.java program. Subsequently, run QueryExecutor.java program to check the output. We have received correct result through QueryExecutor.java with ten employees' names as well as their correct relations. This can prove that the store deletion works and it cleared all triples from the store by taking away the whole rdf store.

To avoid complications, an rdf store and an rdf graph are not the same thing in RDF technology, eventhough we have verified store deletion in this process. A store can contain multiple graphs and must have a default graph. For the sake of simplicity, all the triples in the tutorial are inserted into the default graph. But the triples can be placed in different graphs as well, it all depends on how the store is modelled.

Above is the verification of updating data in the staffing store, includes insert triples, delete triples and delete graphs. The result of this verification proved that in DB2, it allows us to insert triples to the staffing store as well as deleting the triples from the store. It proved the concept of updating rdf data of the store in the tutorial.

5.4 Administration and migration

The tutorial has a brief instruction of how to migrate and administrate an RDF store at the end of the paper. According to the tutorial, the user is able to administrate an RDF stores by collecting the distribution statistics of the store. In Chapter 4, a detailed instruction is placed on this subject based on the combination of the tutorial and the experiment. Like mentioned in the tutorial, there are two ways of statistics collection in RDF store: automated collection and manually collection. In Chapter 4, we have successfully set up the statistics collection in both ways. The statistics about the data in the

RDF store is stored in the table Basic Statistics. (IBM 2013) In order to verify the collected statistics, we have queried the following tables in DB2 database:

`staffing_basestats`
`staffing_topkstats`

Get into db2 command prompt with command `db2` connect to the database "RDFSAMPL" and write `SELECT` statement: `SELECT * FROM staffing_basestats`, similarly, to view the content of the table topkstats: `SELECT * FROM staffing_topkstats`.

There are certain rules, which govern when the statistic collection is needed in case of DB2RDF (Sahoo, 2013):

1. The minimum count of triple in the store should be in excess of 1 Million.
2. The count of triples should increase or decrease by 25% from the previous run.

We have tested this function both manually and automatically and the result was positive.

The migration test could not be completed due to inconsistencies with several jar files. However, theoretically it should work. In Chapter 4, we saw that there are at least two ways of implementing this function: through a java program or the use of a bash file with the corresponding command. We will move the migration testing to our future studies.

5.5 Improvements

During the implementation of the tutorial, we have made one mistake while loading data into the RDF application. On tutorial page 12, after adding the data into the rdf store staffing, we received the expected dataset with a different sequense. We ran the InsertGraph2.java program several times, with the aim to emulate the result as shown in the tutorial. However, this action brought us a large dataset to the DB2 database, because each time when you run the InsertGraph2.java, it adds the same dataset into the database repeately. Hence, after running the InsertGraph2.java for several times,

the volum of the dataset is multipled. So in this tutorial, the data are not unique. For example a exployee whoes Id is 000001 can be inserted 10 times and the information of this same person will be stored in the database as 10 single entries with the same Id number. When we retrive the information of this employee, it will search the database with employee Id and return us a 10 times multipled result. When the volum of the data becomes large in the database, it increases the cost of storage as well as the time of fetching data.

6 Conclusion

The purpose of this study was twofold. The first part of the study was to implement the RDF application in DB2 Express-C on a Linux based operation system. The second part of the study aimed to verify the concept of the used tutorial. In the first part, we have implemented the application based on a tutorial published by IBM. The aim of the implementation was to first go through the tutorial and get familiar with the technology; find out how to implement such an application in DB2, sort out the difficulties, problems and provide corresponding solutions. In the second part, we verified the concept of each RDF function mentioned in the tutorial. Based on the experiment, we proved that the sample RDF application works in DB2.

6.1 Study result

This study has been carried out fully as the proof of concept. The material that we have used for implementation is the tutorial published by IBM: "Resource description framework application development in DB2 for Linux, UNIX and Windows". In the tutorial, the authors have used Windows as their operation system. Hence, some of the commands that we have used are different from the tutorial. These changes are recorded in text in the Chapter 4. Similary in the same chapter, we have listed difficulties and problems that we met during the implementation. For each problem, we corresponded with detailed solution with unclear commands fixed.

In Chapter 5, we have proved four things: Loading data into the RDF store, executing SPARQL queries over HTTP, data updating in RDF store and store administration. First, we examined loading data into the RDF store through a java program. By running this java program, we inserted data from the four simulated systems (Org System, Project System, HR System, Legal System) which were given by the tutorial. The dataset output from the sample RDF store in our system matches the result on the tutorial. Second, we proved that executing SPARQL queries over HTTP is successful in DB2. Some minor changes have been made during this process against the tutorial. We have tested this execution through three different queries. Third, we verified

function of updating data in the RDF store. The test of data updating includes triples instertion and deletion as well as graphs insertion. In the same section, we also proved the concept of store deletion due to a large repeated dataset that we have received. Last but not least, we have tested administration of the RDF store. The test proved that in an RDF store, one is able to collect data either automatically or manually.

6.2 Future studies

Mentioned earlier in Chapter 5, we did not test the deleting of graphs in the RDF store due to the time limit, neither tested the migration due to the inconsistency of several jar files in the rdf folder. These two tests will be covered in our future studies. In addition, we found that there is no unique constraint in the RDF database in this tutorial. The next step of the study is to study about the constraints in RDF: is it possible to add unique contraints in the RDF store? And how to add a unique constraints ? As this study focuses mainly on the implementation of RDF in DB2, we did not cover the related security issues in this study.

7 References

Prud'hommeaux, Eric , Andy Seaborne, and Hewlett-Packard Laboratories. "SPARQL Query Language for RDF." *W3C*. January 15, 2008. http://www.w3.org/TR/rdf-sparql-query/ (accessed February 20, 2013).

www.w3.org. *SPARQL Query Language for RDF*. 03 26, 2013. http://www.w3.org/TR/rdf-sparql-query/ (accessed 03 28, 2013).

W3.org. *N-Triples, W3C RDF Core WG Internal Working Draft*. September 6, 2001. http://www.w3.org/2001/sw/RDFCore/ntriples/ (accessed March 20, 2013).

W3C. *SPARQL 1.1 Update*. March 21, 2013. http://www.w3.org/TR/sparql11-update/ (accessed April 10, 2013).

—. "Resource Description Framework (RDF): Concepts and Abstract Syntax." *W3C Recommendation*. 10. February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#section-data-model (accessed 26. February 2013).

w3schools.com. *Introduction to OWL*. unknown. http://www.w3schools.com/rdf/rdf_owl.asp (accessed March 20, 2013).

World Wide Web Consortium. "W3C Semantic Web Activity." *W3C*. March 29, 2013. http://www.w3.org/2001/sw/#rdf (accessed April 10, 2013).

Virtualbox.org. *Chapter 1. First steps*. unknown, unknown, unknown. https://www.virtualbox.org/manual/ch01.html (accessed 26, February, 2013).

Adams, Nico. *what is an rdf triple?* August 25, 2012. http://stackoverflow.com/questions/273218/what-is-an-rdf-triple (accessed 18, March, 2013).

Balani, Naveen. "The future of the Web is Semantic ." *IBM developerWorks*. October 18, 2005. http://www.ibm.com/developerworks/web/library/wa-semweb/#6 (accessed 18, March, 2013)

Explorable.com. *Understanding on how to conduct science experiments is crucial for understanding how knowledge is created*. 6. November 2011. http://explorable.com/conduct-science-experiments (accessed 15. April 2013).

DB2onCampus. *DB2 on Campus*. unknow unknow, 2008. http://db2oncampus.com/.
IBM. "Authorization, privileges, and object ownership." *DB2 solution Information Center home. (accessed* January 25, 2013.)

http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.sec.doc%2Fdoc%2Fc0005478.html (accessed February 25, 2013).

—. "Automatic table maintenance in DB2, Part 1: Automatic statistics collection in DB2 for Linux, UNIX, and Windows." *IBM developerworks*. June 7, 2007. http://www.ibm.com/developerworks/data/library/techarticle/dm-0706tang/ (accessed March 20, 2013).

—. "DB2 Basics: Getting to know the DB2 UDB command line processor." *IBM developerWorks*. March 10, 2005. http://www.ibm.com/developerworks/data/library/techarticle/dm-0503melnyk/ (accessed April 6, 2013).

—. "DEACTIVATE DATABASE Command." *DB2 Information Center*. 2008. http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/core/r0002039.htm (accessed Feb 20, 2013).

—. ”RDF store tables.” *DB2 Information Center home.* 16. March 2013. http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.swg.im.dbclient.rdf.doc%2Fdoc%2Fr0060049.html (accessed 10. April 2013).

—. "Technical Library." *IBM - developerWorks.* May 24, 2012. http://www.ibm.com/developerworks/views/data/libraryview.jsp?search_by=Resource+description+framework+application+development (accessed Jan 20, 2013).

—. "The administrative task scheduler." *IBM.* unknow. http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.gui.doc/doc/c0054380.html (accessed April 6, 2013).

H, Tarvainen. 14. Jun 2003. http://web.lib.hse.fi/Fi/yrityspalvelin/pdf/2001/Ealdatas.pdf (accessed 20. February 2013).

Jena.apache.org. *Apache Jena.* April unknown, 2012. http://jena.apache.org (accessed April 10, 2013).

Joseki. *Joseki Configuration.* unknow unknow, 2010. http://joseki.sourceforge.net/configuration.html (accessed March 17, 2013).

Joseki.org. *Joseki - A SPARQL Server for Jena.* 7 10, 2010. http://www.joseki.org (accessed April 8, 2013).

Patrick , Hayes. "RDF Semantics." *W3C Recommendation.* unknown unknown, 2004. http://www.w3.org/TR/rdf-mt/#graphdefs (accessed March 20, 2013).

Sahoo, Priya Ranjan, haastattelu, haastattelijana Yuqing He. *Senior Software Engineer in IBM* (16. April 2013).

RDF Working Group. "Resource Description Framework (RDF)." *W3C Semantic Web.* February 10, 2004. http://www.w3.org/RDF/ (accessed February 20, 2013).

Appendix:

InsertGraph2.java

```
/*
* Licensed Materials - Property of IBM
* 'Sample application for developerWorks article on
* RDF Application Development Tutorial ? Part 1'
*
* ? Copyright IBM Corporation 2012. All Rights Reserved.
*
* The source code for this program is not published or otherwise divested of
* its trade secrets, irrespective of what has been deposited with the U.S.
* Copyright Office.
* Note to U.S. Government Users Restricted Rights:
* Use, duplication or disclosure restricted by GSA ADP
* Schedule Contract with IBM Corp.
*/
package com.ibm.db2.rdf.sample;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.openjena.riot.Lang;
import org.openjena.riot.RiotLoader;
import org.openjena.riot.RiotWriter;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.sparql.core.DatasetGraph;
import com.hp.hpl.jena.sparql.core.DatasetGraphFactory;
import com.hp.hpl.jena.util.FileManager;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
public class InsertGraph2 {
public static void main(String[] args) throws SQLException {
Connection conn = null;
Store store = null;
```

```
String storeName = "staffing";
String schema = "db2inst1";

try {
Class.forName("com.ibm.db2.jcc.DB2Driver");
conn = DriverManager.getConnection(
"jdbc:db2://localhost:50001/RDFSAMPL", "db2inst1", "password");
conn.setAutoCommit(false);
} catch (ClassNotFoundException e1) {
e1.printStackTrace();
}


// Delete store if required.
//StoreManager.deleteStore(conn, schema, storeName);

// Create a store or dataset.
store = StoreManager.createStore(conn, schema, storeName, null);

//Connect to the store
store = StoreManager.connectStore(conn, schema, storeName);
conn.commit();

//Create a store or dataset.
//store = StoreManager.createStore(conn, schema, storeName, null);


/*
 * Generally keep the 'Store' object hanging around. Otherwise there is
 * always an unnecessary query to know which set of tables we need to
 * work with. - The Store object does not keep a reference to the
 * connection passed to the StoreManager methods. Thats why in the API u
 * need to pass a connection again in RDFStoreFactory's methods. - It is
 * OK to use all other objects (Dataset/Graph/Model) as lightweight i.e.
 * Create afresh for each request.
 */

// create an in in-memory model for each of the individual  n-triple files
Model mHr = createInMemoryModel("./resources/hr.nt");
```

```
Model mProj = createInMemoryModel("./resources/project.nt");
Model mOrg = createInMemoryModel("./resources/org.nt");
Model mLegal = createInMemoryModel("./resources/legal.nt");

// merge all the four in-memory models into a single graph
Model merged = mHr.union(mProj);
merged = merged.union(mOrg);
merged= merged.union(mLegal);

// connect to the defaultModel in the staffing store
Model staffingDefaultModel = RdfStoreFactory.connectDefaultModel(store, conn);
staffingDefaultModel.begin();

// add the merged in-memory model to the staffing model

staffingDefaultModel.add(merged);
staffingDefaultModel.commit();

//Verify the data is stored in DB2 by dumping out the data from DB2.
Dataset ds = RdfStoreFactory.connectDataset(store, conn);
RiotWriter.writeNQuads(System.out, ds.asDatasetGraph());
}

private static Model createInMemoryModel(String inputFileName) {
Model memModel = null;

// use the FileManager to find the input file
InputStream in = FileManager.get().open(inputFileName);
if (in == null) {
   throw new IllegalArgumentException(
                           "File: " + inputFileName + " not found");
}

// read the RDF/XML file
DatasetGraph dsg = DatasetGraphFactory.createMem();
// since ntriples is nothing but n-quad without graph, this will
// just go to default graph
RiotLoader.read(in, dsg, Lang.NQUADS, "");
memModel = ModelFactory.createModelForGraph(dsg.getDefaultGraph());
```

```
try {
                in.close();
} catch (IOException e) {
                e.printStackTrace();
}
return memModel;
}
}
```

QueryExcecutor

```
 * Licensed Materials - Property of IBM
 * 'Sample application for developerWorks article on
 *  RDF Application Development Tutorial – Part 1'
 * – Copyright IBM Corporation 2012. All Rights Reserved.
 * The source code for this program is not published or otherwise divested of
 * its trade secrets, irrespective of what has been deposited with the U.S.
 * Copyright Office.
 * Note to U.S. Government Users Restricted Rights:
 * Use, duplication or disclosure restricted by GSA ADP
 * Schedule Contract with IBM Corp.
 */
package com.ibm.db2.rdf.sample;

//import
com.ibm.rdf.store.cmd.DumpRdfStore("/home/db2inst1/Desktop/Export.nq");
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.openjena.riot.RiotWriter;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.rdf.model.Model;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryExecutionFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryFactory;
public class QueryExcecutor {
        public static void main(String[] args) throws SQLException
        {
                Connection conn = null;
                Store store = null;
```

```
                String storeName = "staffing";
                String schema = "db2inst1";
                try {
                        Class.forName("com.ibm.db2.jcc.DB2Driver");
                        conn = DriverManager.getConnection(
                "jdbc:db2://localhost:50001/RDFSAMPL", "db2inst1", "pass-
word");
                        conn.setAutoCommit(false);
                } catch (ClassNotFoundException e1) {
                        e1.printStackTrace();
                }
                // Connect to the store
                store = StoreManager.connectStore(conn, schema, store-
Name);

/*
 * Generally keep the 'Store' object hanging around. Otherwise there is
 * always an unnecessary query to know which set of tables we need to
 * work with. - The Store object does not keep a reference to the
 * connection passed to the StoreManager methods. Thats why in the API u
 * need to pass a connection again in RDFStoreFactory's methods. - It is
 * OK to use all other objects (Dataset/Graph/Model) as lightweight i.e.
 * Create afresh for each request.
 */
                Dataset ds = RdfStoreFactory.connectDataset(store, conn);
                RiotWriter.writeNQuads(System.out, ds.asDatasetGraph());
                printAllEmployeeFriends(store, conn);
        }
        private static void printAllEmployeeFriends(Store store, Connection conn)
{
                String query = "select ?member ?friendname where { " +
                "<http://xyz.com/project/robotX>
<http://xyz.com/project/member> ?memberId . " +
                                                "?memberId
<http://xmlns.com/foaf/0.1/name> ?member . " +
                                                "?memberId
<http://xmlns.com/foaf/0.1/knows> ?friendId . " +
                                                " ?friendId
<http://xmlns.com/foaf/0.1/name>  ?friendname " +
```

```
                                                    "}";
                        Query q = RdfStoreQueryFactory.create(query);
                        Dataset ds = RdfStoreFactory.connectDataset(store, conn);
                        QueryExecution qe = RdfStoreQueryExecutionFacto-
ry.create(q, ds);
                        Model m = null;
                        if (q.isSelectType()) {
                                        ResultSet rs = qe.execSelect();
                                        while (rs.hasNext()) {
                                           QuerySolution qs = rs.next();
                                           System.out.println(qs);
                                           System.out.println();
                                        }
                        } else if (q.isDescribeType()) {
                                        m = qe.execDescribe();
                                        m.write(System.out, "N-TRIPLE");
                        } else if (q.isAskType()) {
                                        System.out.println(qe.execAsk());
                        } else if (q.isConstructType()) {
                                        m = qe.execConstruct();
                                        m.write(System.out, "N-TRIPLE");
                        }
                        qe.close();
                        if (m != null) {
                                        System.out.println("Number of Rows  : " +
m.size());                              m.close();
                        }
                }
}
```

```
* Licensed Materials - Property of IBM
 * 'Sample application for developerWorks article on
 *  RDF Application Development Tutorial – Part 1'
 * – Copyright IBM Corporation 2012. All Rights Reserved.
 *
 * The source code for this program is not published or otherwise divested of
 * its trade secrets, irrespective of what has been deposited with the U.S.
 * Copyright Office.
 * Note to U.S. Government Users Restricted Rights:
 * Use, duplication or disclosure restricted by GSA ADP
 * Schedule Contract with IBM Corp.
 */

package com.ibm.db2.rdf.sample;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.openjena.riot.Lang;
import org.openjena.riot.RiotLoader;
import org.openjena.riot.RiotWriter;

import com.hp.hpl.jena.graph.Graph;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.sparql.core.DatasetGraph;
import com.hp.hpl.jena.sparql.core.DatasetGraphFactory;
import com.hp.hpl.jena.util.FileManager;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
public class InsertTriple {
```

```
        public static void main(String[] args) throws SQLException {
                Connection conn = null;
                Store store = null;
                String storeName = "staffing";
                String schema = "db2inst1";
                try {
                        Class.forName("com.ibm.db2.jcc.DB2Driver");
                        conn = DriverManager.getConnection(
                        "jdbc:db2://localhost:50001/RDFSAMPL",
"db2inst1", "password");
                        conn.setAutoCommit(false);
                } catch (ClassNotFoundException e1) {
                        e1.printStackTrace();

                }

// Create a store or dataset.
//store = StoreManager.createStore(conn, schema, storeName, null);
// Connect to the store

store = StoreManager.connectStore(conn, schema, storeName);

/*
 * Generally keep the 'Store' object hanging around. Otherwise there is
 * always an unnecessary query to know which set of tables we need to
 * work with. - The Store object does not keep a reference to the
 * connection passed to the StoreManager methods. Thats why in the API u
 * need to pass a connection again in RDFStoreFactory's methods. - It is
 * OK to use all other objects (Dataset/Graph/Model) as lightweight i.e.
 * Create afresh for each request.
 */

// get the new triples to be added
//StmtIterator it = getNewTriplesToAdd("./resources/newdata.nt");
                StmtIterator it = getNewTri-
plesToAdd("/home/db2inst1/Desktop/DB2RDFTutorial/resources/newdata.nt");

// connect to the defaultModel in the store
```

```
Model storeDefModel = RdfStoreFactory.connectDefaultModel(store, conn);

                        // add each new triple to the store

                        storeDefModel.begin();
                        while (it.hasNext()) {
                                        storeDefModel.add(it.next());
                        }
                        storeDefModel.commit();

        }
        private static StmtIterator getNewTriplesToAdd(String inputFileName) {
// use the FileManager to find the input file
                        InputStream in = FileManager.get().open(inputFileName);
                        if (in == null) {
                           throw new IllegalArgumentException(
                                                "File: " + inputFileName + " not
found");
                        }
// read the RDF/XML file
                        DatasetGraph dsg = DatasetGraphFactory.createMem();

// since ntriples is nothing but nquad without graph, this will
// just go to defaultgraph
                        RiotLoader.read(in, dsg, Lang.NQUADS, "");
                        Model memModel = ModelFacto-
ry.createModelForGraph(dsg.getDefaultGraph());
                        try {
                                        in.close();
                        } catch (IOException e) {
                                        e.printStackTrace();
                        }
                        return memModel.listStatements();
        }

}
```

```
/*
 * Licensed Materials - Property of IBM
 * 'Sample application for developerWorks article on
 *  RDF Application Development Tutorial – Part 1'
 * – Copyright IBM Corporation 2012. All Rights Reserved.
 * The source code for this program is not published or otherwise divested of
 * its trade secrets, irrespective of what has been deposited with the U.S.
 * Copyright Office.
 * Note to U.S. Government Users Restricted Rights:
 * Use, duplication or disclosure restricted by GSA ADP
 * Schedule Contract with IBM Corp.
 */

package com.ibm.db2.rdf.sample;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import com.hp.hpl.jena.graph.Graph;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.impl.ResourceImpl;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
public class DeleteTriple {
        public static void main(String[] args) throws SQLException {
                Connection conn = null;
                Store store = null;
                String storeName = "staffing";
                String schema = "db2admin";
                try {
```

```
                                        Class.forName("com.ibm.db2.jcc.DB2Driver");
                                        Properties prop = new Properties();
                                        prop.setProperty("enableExtendedIndicators",
"2");
                                        prop.setProperty("user", "db2admin");
                                        prop.setProperty("password", "db2admin");
                                        conn = DriverManager.getConnection(
                                "jdbc:db2://localhost:50000/RDFSAMPL", prop);

                                        conn.setAutoCommit(false);



                                } catch (ClassNotFoundException e1) {
                                        e1.printStackTrace();
                                }

// Connect to the store
                                store = StoreManager.connectStore(conn, schema, store-
Name);



/*
 * Generally keep the 'Store' object hanging around. Otherwise there is
 * always an unnecessary query to know which set of tables we need to
 * work with. - The Store object does not keep a reference to the
 * connection passed to the StoreManager methods. Thats why in the API u
 * need to pass a connection again in RDFStoreFactory's methods. - It is
 * OK to use all other objects (Dataset/Graph/Model) as lightweight i.e.
 * Create afresh for each request.
 */

//Remove a triple via JENA Model interface
                                removeEmployeeInformationUsingModelInterface(store,
conn, "000001");

//Remove a triple via JENA Graph interface
```

```
                removeEmployeeInformationUsingGraphInterface(store,
conn, "000002");
        }
        public static void removeEmployeeInformationUsingModelInterface(Store
store, Connection conn, String eid) throws SQLException {
                //Connect to the default model
                Model defModel = RdfStoreFacto-
ry.connectDefaultModel(store, conn);
                //List the statements for the given subject
                Resource employee = new Resource-
Impl("http://xyz.com/employee#"+eid);
                StmtIterator stmtIterator = defMod-
el.listStatements(employee,null,(RDFNode)null);
                defModel.begin();

//Remove the statements for the given subject
                defModel.remove(stmtIterator);

                defModel.commit();
        }
        public static void removeEmployeeInformationUsingGraphInterface(Store
store, Connection conn, String eid) throws SQLException {
                //Connect to the default graph
                Graph defGraph = RdfStoreFacto-
ry.connectDefaultGraph(store, conn);
                //Create the triple to be matched
                Node s =
Node.createURI("http://xyz.com/employee#"+eid);
                Triple triple = new Triple(s, Node.ANY, Node.ANY);
                //Find and delete all the matching triples
                ExtendedIterator<Triple> matchedTripleIterator =
defGraph.find(triple);
                defGraph.getTransactionHandler().begin();
                while(matchedTripleIterator.hasNext())
                {
                        Triple matchedTriple = matchedTripleItera-
tor.next();
                        defGraph.delete(matchedTriple);
                }
```

```
                defGraph.getTransactionHandler().commit();
            }

}
```

DeleteGraph

```
/*
 * Licensed Materials - Property of IBM
 * 'Sample application for developerWorks article on
 *  RDF Application Development Tutorial – Part 1'
 * – Copyright IBM Corporation 2012. All Rights Reserved.
 * The source code for this program is not published or otherwise divested of
 * its trade secrets, irrespective of what has been deposited with the U.S.
 * Copyright Office.
 * Note to U.S. Government Users Restricted Rights:
 * Use, duplication or disclosure restricted by GSA ADP
 * Schedule Contract with IBM Corp.
 */
package com.ibm.db2.rdf.sample;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import com.hp.hpl.jena.graph.Graph;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Triple;
import com.hp.hpl.jena.sparql.core.DatasetGraph;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
public class DeleteGraph {
        public static void main(String[] args) throws SQLException {
                Connection conn = null;
                Store store = null;
                String storeName = "staffing";
                String schema = "db2admin";
                try {
        Class.forName("com.ibm.db2.jcc.DB2Driver");
conn = DriverManager.getConnection("jdbc:db2://localhost:50000/RDFSAMPL",
"db2admin", "db2admin");
                        conn.setAutoCommit(false);
                } catch (ClassNotFoundException e1) {

e1.printStackTrace();
```

```
                        }

// Connect to the store
store = StoreManager.connectStore(conn, schema, storeName);

/*
 * Generally keep the 'Store' object hanging around. Otherwise there is
 * always an unnecessary query to know which set of tables we need to
 * work with. - The Store object does not keep a reference to the
 * connection passed to the StoreManager methods. Thats why in the API u
 * need to pass a connection again in RDFStoreFactory's methods. - It is
 * OK to use all other objects (Dataset/Graph/Model) as lightweight i.e.
 * Create afresh for each request.
 */
                        String graphURI = "http://staffing/hr";
//Create a named graph for testing purpose
                        Graph graph = RdfStoreFactory.connectNamedGraph(store,
conn, graphURI);
                        Node s = Node.createURI("http://xyz.com/subject");
                        Node p = Node.createURI("http://xyz.com/predicate");
                        Node v = Node.createURI("http://xyz.com/value");
                        graph.add(new Triple(s, p, v));
                        graph.close();
                        conn.commit();
                        verifyIfGraphIsPresent(conn, store, graphURI);
// Remove a named graph from the store.
                        removeNamedGraph(store, conn, graphURI);
                        conn.commit();
                        verifyIfGraphIsPresent(conn, store, graphURI);
                }

                private static void verifyIfGraphIsPresent(Connection conn, Store
store,String graphURI) {

// Verify that the hr graph is not existing

DatasetGraph dsg = RdfStoreFactory.connectDataset(store, conn).asDatasetGraph();
```

```
System.out.println("Is the graph "+graphURI+" present in the database ? " +
dsg.containsGraph(Node  .createURI(graphURI)));

            }
public static void removeNamedGraph(Store store, Connection conn, String graphURI)
{

DatasetGraph dsg = RdfStoreFactory.connectDataset(store, conn).asDatasetGraph();
            dsg.removeGraph(Node.createURI(graphURI));
            dsg.close();
            }

}
```

Dowloads

Description	Name	Size	Download method
VirtualBox	VirtualBox 4.2.6 for OS X hosts	14.9 MB	Download
DB2RDF Javadoc	DB2RDFjavadoc.zip	78 KB	HTTP
Tutorial samples	DB2RDFTutorial.zip	21 KB	HTTP
OpenJENA	Jena-2.6.3.jar	1.9 MB	Binary Download
ARQ	Arq-2.8.8	315 KB	Binary Download
JOSEKI	Joseki-3.4.4.zip	13.3 MB	Download