IBM DB2 10.5
for Linux, UNIX, and Windows

Developing RDF Applications for IBM Data Servers

SC27-5505-00

IBM DB2 10.5
for Linux, UNIX, and Windows

Developing RDF Applications for IBM Data Servers

SC27-5505-00

Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 69.

Contents

Part 1. RDF application development for IBM data servers 1

Chapter 1. RDF references and related resources 3

Chapter 2. RDF store tables 5
RDF administrative database objects 5

Chapter 3. Access control for RDF stores 7

Chapter 4. Default and optimized RDF stores 9

Chapter 5. Central view of RDF stores 11

Chapter 6. Setting up an RDF environment 13
RDF with DB2 Version 9.7 14

Chapter 7. Creating an RDF store . . . 15
Creating a default RDF store 15
Creating an optimized RDF store 16
Creating an optimized RDF store by using APIs 16
Creating an optimized RDF store by using commands. 18
Creating an optimized RDF store with existing data 18
Creating an RDF store using graph-level access control 20

Chapter 8. Modifying data in an RDF store 23
Modifying data in an RDF store 23
SPARQL UPDATE support 25
SPARQL graph update 25
SPARQL graph management. 26
Modifying an RDF store by using SPARQL UPDATE APIs 26

Chapter 9. Querying an RDF store . . . 29
RDF queries and API 29
SPARQL query support 29
JENA model API support. 30
Issuing SPARQL queries 31
Creating a union of all named graphs 33
Registering custom DESCRIBE handlers 34
Enforcing graph level access control using DB2 database server 37
Enforcing graph level access control by using the RDF store SQL generator 37

Chapter 10. Setting up SPARQL Version 1.1 Graph Store Protocol and SPARQL over HTTP 41

Chapter 11. Maintaining an RDF store 43
Updating statistics in an RDF store 43
Converting a default store to an optimized RDF store. 44
Verifying if an RDF store needs reorganizing . . 44
Creating reorganized tables for an RDF store . . 44
Switching to reorganized tables in an RDF store 45

Chapter 12. RDF commands 47
createrdfstore command 47
createrdfstoreandloader command 48
droprdfstore command 50
genpredicatemappings command 51
loadrdfstore command. 52
queryrdfstore command 53
reorgcheckrdfstore command 54
reorgrdfstore command 55
reorgswitchrdfstore command 57
setstatsschedule command 58
updaterdfstorestats command 59

Part 2. Appendixes. 61

Appendix A. Overview of the DB2 technical information 63
DB2 technical library in hardcopy or PDF format . . 63
Displaying SQL state help from the command line processor 66
Accessing different versions of the DB2 Information Center 66
Terms and conditions 66

Appendix B. Notices 69

Index 73

Part 1. RDF application development for IBM data servers

The Resource Description Framework (RDF) is a family of W3 specifications that you can use as a standard data interchange framework for modeling information. Applications can store and query RDF data in IBM® DB2® 10.5 Enterprise Server Edition (DB2 Enterprise Server Edition) databases.

RDF employs Uniform Resource Identifiers (URIs) to create a relationship between data as a triple, for example, in the form of subject-predicate-object expressions. You can link, expose, and share structured and semi-structured data across different applications by using this simple model.

An RDF store in the DB2 database server is a set of user tables within a database schema that stores an RDF data set. A unique store name is associated with each set of these tables. Each RDF store has a table that contains metadata for the store. This table has the same name as the store.

You can load data into the user tables by using RDF utility commands or Java™ APIs. You must have the appropriate read and write permissions on these sets of tables. The Java APIs that are supported are the JENA framework APIs. DB2 RDF utility commands or APIs are supported for the DB2 software Version 9.7 and later.

RDF applications use the SPARQL query language to retrieve data in DB2 databases.

RDF is not supported in partitioned database environments.

Chapter 1. RDF references and related resources

Many resources are available to help you develop RDF applications that access IBM data servers.

Table 1. RDF references and related resources

RDF resource	Reference Link
RDF Primer	http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
SPARQL query language	http://www.w3.org/TR/rdf-sparql-query/
JENA graph and model APIs	http://jena.sourceforge.net/tutorial/RDF_API/
IBM RDF Javadoc	../javadoc/index.html
RDF application development tutorial Part 1: RDF store creation and maintenance	http://www.ibm.com/developerworks/data/tutorials/dm-1205rdfdb210/index.html

Chapter 2. RDF store tables

An RDF store consists of multiple tables. These tables contain metadata about the RDF store or user data.

Metadata tables

The following tables contain metadata about the RDF store:

- One metadata table, which has the same name as the RDF store.
- The System Predicates metadata table, which stores information about RDF predicates that you can use for further filtering results of a SPARQL query.
- The Basic Statistics table, which stores statistics about the data in the RDF store.
- The Top K Statistics table, which stores information about the least selective RDF data in the store.

Data tables

The following tables store RDF data if the value of the data does not exceed a particular character length:

- The Direct Primary table stores RDF triples and the associated graph, indexed by subject. Predicates and objects for a subject are stored in pair columns in this table. A particular predicate can occur in any one of three columns in this table. The object for that predicate is stored in the corresponding object column of the predicate-object pair.
- The Direct Secondary table stores RDF triples that share the subject and predicate within an RDF graph. Such triples have only a placeholder identifier in the Direct Primary table.
- The Reverse Primary table stores RDF triples and the associated graph, indexed by object. Predicates and subjects for an object are stored in pair columns in this table. A particular predicate can occur in any one of three columns in this table and the subject for that predicate can occur in the corresponding subject column of that pair.
- The Reverse Secondary table stores RDF triples that share the object and predicate within an RDF graph. Such triples have a placeholder identifier in the Reverse Primary table.
- The Datatypes table stores the mapping of internal integer values for SPARQL data types, user-defined data types, and language tags.

If the value of an RDF subject, predicate, object, or graph exceeds a particular character length, the previously mentioned five tables store a placeholder identifier only. The actual value is stored in the Long Strings table.

RDF administrative database objects

RDF has functions and a scheduler task object to manage an RDF store.

Administrative database objects

RDF for DB2 has administrative database objects, as follows:

- A Java based external UDF named <*store_name*>_RDF_REGEX supports the regex operator in SPARQL. Appropriate permission must be granted to use this UDF.

  In DB2 Version 10.1 Fix Pack 2 and later fix packs, <*store_name*>_RDF_REGEX UDF is no longer supported for regular expression functionality. Instead use the pureXML® `fn:matches()` function for regular expression
- An SQL stored procedure named <*store_name*>_T3_STATS gathers the basic and topK statistics for an RDF store.
- An administrative scheduler task named <*SCHEMANAME*>_<*STORENAME*>_Scheduler is used to schedule the interval for the updates of the stores basic and topK statistics.

Chapter 3. Access control for RDF stores

Two types of access control are available for DB2 RDF stores.

Coarse grained access control

You can use DB2 database's table level permissions to control access to the entire RDF store.

RDF graph level access control

RDF graph level access control provides more fine grained access control at the level of RDF graphs. You can selectively control the RDF graphs to which users will have access in the RDF store, rather than the whole RDF data set.

With RDF graph level access control, RDF triples within a graph are used to determine whether a user has access to the RDF graph or not. At the time of RDF store creation, the user needs to specify which RDF predicates will be used to control access to the RDF Graph.

Enforcing access control during runtime (using SPARQL queries) can be delegated to the DB2 engine. Alternatively, it can be used in the SQL generated by the DB2 RDF Store SQL generator.

If you chose that the access control is enforced by the DB2 engine, you need to use the fine Grained access control feature of the DB2 software, to specify the access control rules.

If you chose that the access control is enforced by the RDF Store SQL Generator, the application needs to additionally pass in the constraints to be applied in the `QueryExecution` context. In this case only a limited set of operators and operand are supported:

- Creating an RDF Store with Graph level Access Control support
- Enforcing Graph level Access Control via the RDF Store SQL Generator
- Enforcing Graph level Access Control via the DB2 engine

Chapter 4. Default and optimized RDF stores

Two kinds of RDF stores are used with DB2 databases. One is referred to as default RDF store, while the other is referred to as an optimized RDF store.

Default RDF stores

This base schema is used when nothing is known about the RDF data being stored or no appropriate sample is available. Default RDF stores use a default number of columns in the Direct Primary and Reverse Primary tables. You use the default store when starting with a new RDF data set, about which nothing is known. In the default store hashing is used to determine the columns to which the predicates and objects go to in the Direct Primary and Reverse Primary tables.

Create a default RDF store when you have no existing sample data of the RDF data set on which predicate coexistence can be calculated by the DB2 software.

Optimized RDF stores

If sufficient representative data of the RDF data set is already available, then a more optimized schema can be created for the Direct Primary and Reverse Primary tables. This optimized schema is achieved by exploiting the fact that RDF predicates correlate. For example age and social security number coexist as predicates of Person, and headquarters and revenue coexist as predicates of Company, but age and revenue never occur together in any entity.

Create a optimized RDF store when you have existing or sample data for the RDF data set on which DB2 will calculate predicate correlation to assign predicates to columns intelligently.

Advantages of optimized RDF stores

Predicate correlation is used to drastically, and in many cases completely, remove the randomness of hashing used in default stores. So, in default stores predicate collisions can occur because of lack of knowledge of predicate correlation and this can cause more rows to used in the table than actually required. Extra rows could cause joins between tables to be less efficient than they need be.

Indexing predicates can be more easily achieved since mostly a given predicate can be confined to one single column. Also predicates that don't coexist can be assigned to a single column, allowing a single DB2 index to index multiple predicates.

Chapter 5. Central view of RDF stores

Starting with DB2 Version 10.1 Fix Pack 2 and later fix packs, Resource Description Framework (RDF) now lists all RDF stores that are present in a particular database within one table. Query the SYSTOOLS.RDFSTORES table to view all the RDF stores.

The SYSTOOLS.RDFSTORES table is created the first time that the **createrdfstore** or **createrdfsoreandloader** command and API is issued for a database.

Table 2. SYSTOOLS.RDFSTORES table schema

Column name	Data type	Nullable	Description
STORENAME	VARCHAR(256)	NO	Name of the RDF store
SCHEMANAME	VARCHAR(256)	NO	Name of the schema for the RDF store
STORETABLE	VARCHAR(256)	NO	Name of the metadata table for the RDF store
Primary key		NO	Primary key

To list all the RDF stores and their corresponding schema names in a database, issue the following query:

```
SELECT storeName, schemaName FROM SYSTOOLS.RDFSTORES
```

The following sample output is returned:

```
STORENAME       SCHEMANAME
---------------------------
STAFFING   DB2ADMIN
SAMPLSTORE      DB2ADMIN

2 record(s) selected.
```

Chapter 6. Setting up an RDF environment

Set up your environment to use DB2 RDF command and APIs.

Issuing RDF commands by using command-line utilities

DB2 RDF command-line utilities can be found in the `<install_path>/sqllib/rdf/bin` directory. Start the utilities from this directory with a DB2 Command Prompt.

After the DB2 database server is installed, complete the following tasks to use DB2 RDF command-line utilities:

1. Download ARQ package Version 2.8.5 from http://sourceforge.net/projects/jena/files/ARQ/ARQ-2.8.5/“http://sourceforge.net/projects/jena/files/ARQ/ARQ-2.8.5/”.

   Copy the JAR files from the `lib` folder of the ARQ package to the `<install_path>/SQLLIB/rdf/lib` directory.

   Note: You can skip copying over the 'xxx-tests.jar', 'xxx-sources.jar', 'xxx-test-sources.jar' JAR files.

   Starting with DB2 Version 10.1 Fix Pack 2, use the Apache JENA Version 2.7.3 package from http://archive.apache.org/dist/jena/binaries/“http://www.apache.org/dist/jena/binaries/”.

   Save the JAR files from the `lib` folder of the Apache JENA package to the `<install_path>/SQLLIB/rdf/lib` directory.
2. Download the `Commons-logging-1-0-3.jar` from the Apache Commons project. Place this JAR in the `<install_path>/SQLLIB/rdf/lib` directory.
3. Open a command prompt and go to the `<install_path>/SQLLIB/rdf/bin` directory.
   ```
   cd "<install_path>/SQLLIB/rdf/bin"
   ```
4. Add the db2jcc4.jar DB2 JCC driver in the <install_path>/SQLLIB/java directory to the class path environmental variable, as shown:
   ```
   set classpath=<install_path>\SQLLIB\java\db2jcc4.jar;%classpath%
   ```

Now you can run the DB2 RDF command-line utilities in this command prompt.

DB2 RDF in an application development environment

The DB2 RDF JAR files need to be added to the application class path, along with the following JAR file:

- The JENA dependant JAR files
- Commons-logging-1-0-3.jar file
- The DB2 JCC driver (db2jcc4.jar)

The JAR files are in the `<install_path>/sqllib/rdf/lib` directory. They include the following JAR files:

- rdfstore.jar
- antlr-3.3-java.jar
- wala.jar

RDF with DB2 Version 9.7

You can install DB2 Version 10.1 client and use it with DB2 Version 9.7 database server.

Register the external Java libraries that are required for RDF Support. Registration is done by running the 'rdf/bin/registerrdfudf' script in the DB2 Version 10.1 client. This script must be issued for each DB2 Version 9.7 database in which RDF stores are being created. For example, issue the following command:

```
registerrdfudf <dbname> <username>
```

where <*dbname*> is a cataloged database on the local DB2 client.

Chapter 7. Creating an RDF store

Create a default or an optimized RDF store, based on your application development requirements. You can choose to create an RDF store first, and then load the data later.

Creating a default RDF store

You can create a RDF store without any existing RDF data. This is also known as a default RDF store.

Before you begin

The following prerequisites are required:

- Ensure the database has a minimum page size of 32 KB.
- Ensure the **LOGFILSIZ** database configuration parameter is set to is greater than or equal to 20000.
  ```
  db2 UPDATE DATABASE CONFIGURATION FOR <DB_NAME>
  USING LOGFILSIZ 20000
  ```
- Ensure that the SYSTOOLSPACE table space exists.
  ```
  CREATE TABLESPACE SYSTOOLSPACE IN IBMCATGROUP
  MANAGED BY AUTOMATIC STORAGE EXTENTSIZE 4
  ```
- Ensure that the authorization ID has the following privileges:
  - CREATETAB authority for the selected database schema.
  - CREATE EXTERNAL ROUTINES authority.
  - Update privileges for table SYSTOOLS.ADMINTASKSTATUS.

In addition, also set up the following functionality:

- Set the Administrative Task Scheduler to "YES".
  ```
  db2set DB2_ATS_ENABLE=YES
  ```
- Set the **AUTORUNSTATS** database configuration parameter to "ON".
  ```
  db2 UPDATE DB CONFIG USING AUTO_MAINT ON AUTO_TBL_MAINT ON AUTO_RUNSTATS ON
  ```
- Set the bufferpool to "AUTOMATIC", and assign a good initial size.
  ```
  db2 alter bufferpool IBMDEFAULTBP IMMEDIATE SIZE 15000 AUTOMATIC
  ```

Procedure

Use the following instructions to create a default RDF store:

1. Control the names of database tables and table spaces that will make up the RDF store. Create an `objectNames.props` properties file, containing a list of RDF store data tables and the corresponding names and table spaces your want to assign to the tables.

   The contents of a sample `objectNames.props` properties file are shown in the following example:
   ```
   direct_primary_hash=<user_table_name>, <tablespace>
   direct_secondary=<user_table_name>, <tablespace>
   reverse_primary_hash=<user_table_name>, <tablespace>
   reverse_secondary=<user_table_name>, <tablespace>
   long_strings=<user_table_name>, <tablespace>
   basic_stats=<user_table_name>, <tablespace>
   ```

```
topk_stats=<user_table_name>, <tablespace>
system_predicate=<user_table_name>, <tablespace>
data_type=<user_table_name>, <tablespace>
```

Note: Using the `objectNames.props` file to set table names is optional. However, if you choose not to use this properties file, then system generated names will be used instead.

2. Issue the **`createrdfstore`** command. Determine the database instance and schema in which you want to create the RDF store. Also, decide on a logical name for the store. This name needs to be unique among all RDF stores within the database schema.

   For example, use the following command to create a store named rdfStore1 in database DB1 and schema db2admin for host localhost and port 60000:

   ```
   createrdfstore rdfStore1 -host localhost -port 60000 -db DB1
   -user db2admin -password XXX -schema db2admin
   ```

Creating an optimized RDF store

Creating an optimized RDF store by using APIs

You need to have existing set of representative triple data to create an optimized store. The data is required to ensure that predicate occurrence can be calculated properly.

Before you begin

If you do not have a representative sample set of triple data, then create a default store by using the **`createrdfstore`** command. Use the default RDF store until sufficient representative triple data is collected based on which predicate occurrence can be calculated.

When you have sufficient data, use the generatePredicateMappings method of the StoreManager Java class to calculate predicate correlation of the triples in the default RDF store. Save the PredicateMappings output for this API.

When creating an RDF store for a production environment, create a clean new optimized RDF store by using the createStoreWithPredicateMappings method of the StoreManager Java class. Provide the PredicateMappings output you obtained earlier.

Procedure

To create an optimized RDF store, write a program that performs two key steps:

1. Calculates the predicate correlation of the triples in the RDF store that contains the representative data. To calculate the predicate correlation, use thegeneratePredicateMappings method of the StoreManager Java class. Save the output of the `PredicateMappings` API.
2. Creates the optimized RDF store by using the createStoreWithPredicateMappings method of the StoreManager Java class. As input, provide the `PredicateMappings` output that you obtained earlier.

Example

The following Java program shows how you can create an optimized RDF store:

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import com.ibm.rdf.store.StoreManager;


/**
 * This program demonstrates how to create an optimized RDF store
 * by using the predicate correlation information from an existing
 * store. You can also create optimized RDF stores for production
 * environments after a collecting enough representative data cycle
 *  on a default store.
 */
public class CreateOptimizedStore {

 public static void main(String[] args) throws SQLException,
IOException {

  String currentStoreName = "sample";
  String currentStoreSchema = "db2admin";
  Connection currentStoreConn = null;

  /*
   * Connect to the "currentStore" and generate the predicate
     * correlation for the triples in it.
   */
  try {
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   currentStoreConn = DriverManager.getConnection(
    "jdbc:db2://localhost:50000/dbrdf", "db2admin",
     "db2admin");
   currentStoreConn.setAutoCommit(false);
  } catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }

  /* Specify the file on disk where the predicate
     * correlation will be stored.
     */
  String file = "/predicateMappings.nq";
  BufferedOutputStream predicateMappings = new
BufferedOutputStream(new FileOutputStream(file));

  StoreManager.generatePredicateMappings(currentStoreConn,
    currentStoreSchema, currentStoreName,
predicateMappings);

  predicateMappings.close();

  /**
   * Create an optimized RDF store by using the previously
     * generated predicate correlation information.
   */
  String newOptimizedStoreName = "production";
  String newStoreSchema = "db2admin";
  Connection newOptimizedStoreConn =
DriverManager.getConnection(
    "jdbc:db2://localhost:50000/dbrdf",
"db2admin","db2admin");
  BufferedInputStream inputPredicateMappings =
 new BufferedInputStream(
     new FileInputStream(file));

 StoreManager.createStoreWithPredicateMappings(
newOptimizedStoreConn, newStoreSchema,
newOptimizedStoreName, null, inputPredicateMappings);

   }
}
```

Creating an optimized RDF store by using commands

In DB2 Version 10.1 Fix Pack 2 and later fix packs, you can create an optimized store from a default RDF store by using the RDF commands.

Procedure

To create an optimized RDF store from the command prompt:

1. Create a default store by using the **createrdfstore** command.
   ```
   createrdfstore rdfStore1 -db RDFSAMPL
   -user db2admin -password XXX
   ```
2. Add data to this store using SPARQL UPDATE or JENA APIs. Use this default store to collect a set of triple data that can be used to calculate predicate occurrence.
3. Generate the predicate mappings by using the**genpredicatemappings** command .
   ```
   genPredicateMappings MyStore -db RDFSAMPL -user db2admin
   -password db2admin "C:\MyStore_predicate_mappings.txt"
   ```
4. Create an optimized store by using the**createrdfstore** command by passing the **-predicatemappings** parameter.

   Use the predicates generated in the preceding step as input for the **-predicatemappings** parameter.
   ```
   createrdfstore MyOptimizedStore -db RDFSAMPL
   -user db2admin -password XXX
   -predicatemappings "C:\MyStore_predicate_mappings.txt"
   ```

Results

The optimized stored is created.

Creating an optimized RDF store with existing data

You can create an RDF store using existing RDF data.

Before you begin

The following prerequisites are required:

- Ensure the database has a minimum page size of 32 KB.
- Ensure the **LOGFILSIZ** database configuration parameter is set to is greater than or equal to 20000.
  ```
  db2 UPDATE DATABASE CONFIGURATION FOR <DB_NAME>
  USING LOGFILSIZ 20000
  ```
- Ensure that the SYSTOOLSPACE table space exists.
  ```
  CREATE TABLESPACE SYSTOOLSPACE IN IBMCATGROUP
  MANAGED BY AUTOMATIC STORAGE EXTENTSIZE 4
  ```
- Ensure that the authorization ID has the following privileges:
  - CREATETAB authority for the selected database schema.

- CREATE EXTERNAL ROUTINES authority.
- Update privileges for table SYSTOOLS.ADMINTASKSTATUS.

In addition, also set up the following functionality:

- Set the Administrative Task Scheduler to "YES".
  ```
  db2set DB2_ATS_ENABLE=YES
  ```
- Set the **AUTORUNSTATS** database configuration parameter to "ON".
  ```
  db2 UPDATE DB CONFIG USING AUTO_MAINT ON AUTO_TBL_MAINT ON AUTO_RUNSTATS ON
  ```
- Set the bufferpool to "AUTOMATIC", and assign a good initial size.
  ```
  db2 alter bufferpool IBMDEFAULTBP IMMEDIATE SIZE 15000 AUTOMATIC
  ```

On Windows platforms, the **createrdfStoreAndLoader** command requires the CygWin application. The Gawk utility required for this command is Versions 4.0 or later. The Core utility required for this command is Version 8.14 or later. After installing CygWin add `<CgyWin_install_directory>/bin` to the PATH environment variable. If you don't have CygWin on the path, you will see the following error message displayed when you run the command:

```
'Cannot run program "sh": CreateProcess error=2, The system cannot find the
specified file.'
```

On Windows platforms, the **createrdfStoreAndLoader** command can be invoked either from a CygWin command prompt or default command prompt. When using a CygWin command prompt, all file paths (-rdfdata, -storeloadfile, -storeschemafile, -objectnames) must not include the 'cygdrive' prefix. Instead use normal windows path like 'C:\....'.

If any paths specified contain space in the folder or file name, the whole string should be enclosed within double quotes

Procedure

1. Export the existing data into a n-Quad file.
2. Control the names of database tables and table spaces that will make up the RDF store. Create an `objectNames.props` properties file, containing a list of RDF store data tables, and the corresponding names and table spaces your want to assign to the tables.

   The contents of a sample `objectNames.props` file are shown in the following example:

   ```
   direct_primary_hash=<user_table_name>,<tablespace>
   direct_secondary=<user_table_name>,<tablespace>
   reverse_primary_hash=<user_table_name>,<tablespace>
   reverse_secondary=<user_table_name>,<tablespace>
   long_strings=<user_table_name>,<tablespace>
   basic_stats=<user_table_name>,<tablespace>
   topk_stats=<user_table_name>,<tablespace>
   system_predicate=<user_table_name>,<tablespace>
   data_type=<user_table_name>,<tablespace>
   ```

   Note: Using the `objectNames.props` file to set table names is optional. However, if you choose not to use this properties file, then system generated names will be used instead.
3. Issue the **createrdfstoreandloader** command.

   On Windows this command requires CygWin. Also, the Gawk utility required for this command is version 4.0, while the Core utility is version 8.14 or later.

Determine the database instance and schema in which you want to create the RDF store. Also, decide on a logical name for the RDF store. This name must be unique among all RDF stores within the database schema.

Specify the **objectnames** parameter in your command. If you do not specify the **objectNames** parameter, then system generated object names are used for the tables of the RDF store instead. Ensure the output directory already exists on the file system.

This command creates an optimized RDF store using existing data and also generates DB2 database load files that should be used to load the data into the newly created RDF store. The load files are created based on the output of the **storeloadfile** parameter.

For example, issue the following command to create a store named rdfStore2 in database DB1 and schema db2admin for host localhost and port 60000. Specify `myRdfData.nq` for the RDF data file, and specify load.sql as the name of the generated store loader file.

```
createrdfstoreandloader rdfStore2 -host localhost -port 60000 -db DB1
-user db2admin -password XXX -schema db2admin
-rdfdatafile ./myRdfData.nq -storeloadfile ./rdfLoader/load.sql
```

4. Open a CLP enabled DB2 command prompt window and connect to the database instance and schema in which the RDF store was created.
5. Run the ./rdfLoader/load.sql file.

   This will load the data from this generated file into the RDF store.

   Note: Do not use the **-t** argument when running the SQL script, since the SQL script is generated with newline as the command separator.

Creating an RDF store using graph-level access control

You can create an RDF store that uses graph-level access control.

Before you begin

Determine the RDF predicates whose triples are used to control access to the RDF graph. For example, you can use the ContextId (http://myapp.net/xmlns/CONTEXTID) and AppId (http://myapp.net/xmlns/APPID) predicates. The filters that you use to control access to the graph are also known as *filter predicates*.

Determine the DB2 data types for the RDF object values for these predicates. Currently, only the DB2 VARCHAR data type is supported.

Procedure

To creating an RDF store that uses graph-level access control, write a program that uses the createStore() method of the StoreManager class. This method takes a properties argument. Specify the filter predicates by using the following properties format: `<RDF_PREDICATE> = <DB2_DATATYPE>`.

Example

The following Java program demonstrates how you can use the StoreManager class to create an RDF store with graph-level access control:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import com.ibm.rdf.store.StoreManager;

public class CreateStoreGraphAccessControl {

 /* Access to the graph is controlled based on the
   * following two RDF predicates.
  */
private final static String APPID =
"http://myapp.net/xmlns/APPID";
 private final static String CONTEXTID =
"http://myapp.net/xmlns/CONTEXTID";

 /**
  * The DB2 data type for these predicates is assigned.
  */
 private final static String APPID_DATATYPE = "VARCHAR(60)";
 private final static String CONTEXTID_DATATYPE = "VARCHAR(60)";



/*
  * Create a java.util.properties file that lists these two
   * properties and their data types, where
   * propertyName is the RDF predicate and
   * propertyValue is the data type for the RDF predicate.
  */
 private static Properties filterPredicateProps = new
Properties();
 static {
  filterPredicateProps.setProperty(APPID, APPID_DATATYPE);
  filterPredicateProps.setProperty(CONTEXTID,
CONTEXTID_DATATYPE);
 }


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

  Connection conn = null;

  // Get a connection to the DB2 database.
  try {
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   conn = DriverManager.getConnection(
     "jdbc:db2://localhost:50000/dbrdf",
"db2admin", "db2admin");
  } catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }

  /*
   *  Create the store with the access control predicates.
  */
  StoreManager.createStore(conn, "db2admin",
"SampleAccessControl", null,
   filterPredicateProps);
 }
}
```

Chapter 8. Modifying data in an RDF store

Data in an RDF store can be modified either by using JENA APIs or SPARQL Update operations.

Modifying data in an RDF store

Work with the data in an RDF store using JENA APIs.

Procedure

To modify data in an RDF store, you can use the following sample program.

Modify triples and graphs in an RDF store by using JENA APIs as shown in the following example.

Example

The following program demonstrates how to modify triples and graphs in an RDF store by using JENA APIs:

```
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.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.Resource;
import com.hp.hpl.jena.vocabulary.VCARD;

import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;

public class RDFStoreSampleInsert {

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

  Connection conn = null;
  Store store = null;
  String storeName = "sample";
  String schema = "db2admin";

  try {
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   conn = DriverManager.getConnection(
     "jdbc:db2://localhost:50000/dbrdf", "db2admin",
     "db2admin");
   conn.setAutoCommit(false);
  } catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }


  // Create a store or dataset.
  store = StoreManager.createStore(conn, schema, storeName, null);
  // If the store exists, connect to it.
  //store = StoreManager.connectStore(conn, schema, storeName);

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



  /*
   * Generally, retain the "store" object. Otherwise, you require
     * a query to know which set of tables you must
     * work with. The Store object does not keep a reference to the connection
     * that is passed to the StoreManager methods. Therefore, in the API,
     * you must pass a connection again in RDFStoreFactory's methods. You
     * can use all other objects (such as a dataset, graph, or model)
     * as lightweight. That is, create an object for each request.
   */

  // Add a entire named graph to the store.
  addNamedGraph(store, conn);

  // Remove an entire named graph.
  removeNamedGraph(store, conn);

  // Add a triple to the default graph.
  addTripleToDefaultGraph(store, conn);

  // Add a triple by using the JENA Graph interface.
  addTripleViaGraphInterface(store, conn);


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

  conn.commit();
 }

 public static void addNamedGraph(Store store, Connection conn) {

  // Connect to a NamedModel in the store.
  Model storeModel = RdfStoreFactory.connectNamedModel(store, conn,
    "http://graph1");

  // Create a in-memory model with some data.
  Model m = getMemModelWithSomeTriples();

  // Add the whole graph to rdfstore.
  storeModel.begin();
  storeModel.add(m);
  storeModel.commit();

  storeModel.close();

 }

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

  Model storeModel = RdfStoreFactory.connectNamedModel(store, conn,
    "http://graph1");

  storeModel.begin();
  storeModel.removeAll();
  storeModel.commit();

 }
```

```
 public static void addTripleToDefaultGraph(Store store, Connection conn) {

  Dataset ds = RdfStoreFactory.connectDataset(store, conn);
  Model m = ds.getDefaultModel();

  // Add information by using thye model object.
  m.begin();

  String personURI = "http://somewhere/JohnSmith";
  String fullName = "John Smith";
  Resource johnSmith = m.createResource(personURI);
  johnSmith.addProperty(VCARD.FN, fullName);

  m.commit();
  m.close();
 }

 public static void addTripleViaGraphInterface(Store store, Connection conn) {

  Graph g = RdfStoreFactory.connectNamedGraph(store, conn,
    "http://graph2");

  Node s = Node.createURI("http://sub1");
  Node p = Node.createURI("http://pred1");
  Node v = Node.createLiteral("val1");

  g.add(new Triple(s, p, v));
  g.close();
 }

 private static Model getMemModelWithSomeTriples() {

  Model m = ModelFactory.createDefaultModel();

  Node s = Node.createURI("somesubject");
  Node p = Node.createURI("somepredicate");
  Node v = Node.createURI("AnObject");
  Triple t = Triple.create(s, p, v);
  m.add(m.asStatement(t));

  s = Node.createURI("someothersubject");
  p = Node.createURI("someotherpredicate");
  v = Node.createURI("AnotherObject");
  t = Triple.create(s, p, v);
  m.add(m.asStatement(t));

  return m;
 }
}
```

SPARQL UPDATE support

Starting with DB2 Version 10.1 Fix Pack 2 and later fix packs, SPARQL Version 1.1 UPDATE is supported. SPARQL UPDATE Version 1.1 supports two categories of update operations on a graph store.

SPARQL graph update

In DB2 Version 10.1 Fix Pack 2 and later fix packs, SPARQL graph update commands are supported. These commands facilitate the addition and removal of triples from graphs in a graph store.

The following graph update commands are supported:

INSERT DATA
Adds triples that are specified in the query to the destination graph. Creates a destination graph if it does not exist.

INSERT WHERE
Adds triples by matching the pattern of the `WHERE` condition in the query to the destination graph. Creates a destination graph if it does not exist.

DELETE DATA
Removes triples that are specified in the query. Deleting triples that are not present in an RDF store or a graph has no effect and results in success.

DELETE WHERE
Removes triples by matching the pattern that is specified in the `WHERE` clause of the query. Deleting triples that are not present in an RDF store or a graph has no effect and results in success.

LOAD
Reads an RDF document from an Internationalized Resource Identifier (IRI). and inserts its triples into a specified graph. Creates a destination graph if it does not exist.

CLEAR
Removes all the triples in a specified graph.

SPARQL graph management

In DB2 Version 10.1 Fix Pack 2 and later fix packs, SPARQL graph management commands that create and delete graphs in a graph store are supported. The commands also provide shortcuts for graph update operations, often used during graph management to add, move, and copy graphs.

The following graph management commands are supported:

CREATE
Creates a graph in the graph store. The graph is not persisted because DB2 RDF does not save empty graphs.

DROP Removes specified graphs from the graph store.

COPY Inserts all data from an input graph into a destination graph. Data from the input graph is not affected. Data from the destination graph, if any, is removed before insertion.

MOVE
Moves all data from an input graph into a destination graph. The input graph is removed after insertion. Data from the destination graph, if any, is removed before insertion.

ADD Inserts all data from an input graph into a destination graph. Data from the input graph is not affected. Initial data from the destination graph, if any, is kept intact.

Modifying an RDF store by using SPARQL UPDATE APIs

In DB2 Version 10.1 Fix Pack 2 and later fix packs, you can update data in an RDF data store by using supported UPDATE APIs in SPARQL Version 1.1.

The following program demonstrates how to modify an RDF store by using SPARQL UPDATE APIs.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.sparql.core.Quad;
import com.hp.hpl.jena.sparql.modify.request.QuadDataAcc;
import com.hp.hpl.jena.sparql.modify.request.UpdateDataInsert;
import com.hp.hpl.jena.sparql.util.NodeFactory;
import com.hp.hpl.jena.update.UpdateAction;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.jena.RdfStoreFactory;
/**
* Sample program for using SPARQL Updates
*/
public class RDFStoreUpdateSample {
 public static void main(String[] args) throws SQLException
  // Create the connection
  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/RDFDB",
     "db2admin", "db2admin");
   conn.setAutoCommit(false);  }
  catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }
  // Connect to the store
  store = StoreManager.connectStore(conn, schema, storeName);

  // Create the dataset
  Dataset ds = RdfStoreFactory.connectDataset(store, conn);
  // Update dataset by parsing the SPARQL UPDATE statement
  // updateByParsingSPARQLUpdates(ds);
  // Update dataset by building Update objects
  // updateByBuildingUpdateObjects(ds);
  ds.close();
  conn.commit();
}

/**
* Update by Parsing SPARQL Update
*
* @param ds
* @param graphNode
*/
private static void updateByParsingSPARQLUpdates(Dataset ds) {
 String update = "INSERT DATA
 { GRAPH <http://example/bookStore>
  { <http://example/book1> <http://example.org/ns#price>  100 } }";
 //Execute update via UpdateAction
 UpdateAction.parseExecute(update, ds);
}
/**
* Update by creating Update objects
*
* @param ds
* @param graphNode
*/
 private static void updateByBuildingUpdateObjects(Dataset ds) {
  // Graph node
  Node graphNode = NodeFactory.parseNode("http://example/book2>");
  Node p = NodeFactory.parseNode("<http://example.org/ns#price>");
```

```
   Node o = NodeFactory.parseNode("1000");
   Quad quad = new Quad(graphNode, s, p, o);
   Node s2 = NodeFactory.parseNode("<http://example/book3>");
   Node o2 = NodeFactory.parseNode("2000");
   Quad quad2 = new Quad(graphNode, s2, p, o2);
   //Create quad data to be added to the store
   QuadDataAcc acc = new QuadDataAcc();
   acc.addQuad(quad);
   acc.addQuad(quad2);
   //Create the Update object
   UpdateDataInsert insert = new UpdateDataInsert(acc);
   //Execute the Update via UpdateAction
   UpdateAction.execute(insert, ds);
  }
 }
```

Chapter 9. Querying an RDF store

Use SPARQL to query data in DB2 Resource Description Framework (RDF) stores.

SPARQL for RDF Version 1.0 is supported. In addition, the following subset of features from SPARQL Version 1.1 are supported:

- `AVG`
- `COALESCE`
- `COUNT`
- `GROUP BY`
- `HAVING`
- `MAX`
- `MIN`
- `SELECT` expressions
- `STRSTARTS`
- `STRENDS`
- `SubQueries`
- `SUM`

Starting with DB2 Version 10.1 Fix Pack 2, the following features from SPARQL Version 1.1 are also supported:

- UPDATE support for SPARQL query language.
- Graph store HTTP protocol support for SPARQL query language.

RDF queries and API

The SPARQL query language is used to modify data in DB2 databases, while the JENA framework APIs provide the programming interface. There are some limitations for DB2 RDF stores.

SPARQL query support

There are various syntactic or semantic restrictions and limitations for SPARQL to consider when you work with RDF data.

Limits on lengths of URIs
An RDF implementation of DB2 database can store URIs of any length. However, only the first 2000 characters are used for comparison operations.

Limits on lengths of literals
An RDF implementation of a DB2 database stores literals of any length. However, only the first 2000 characters are used for comparison operations, and other operations such as STRSTARTS and STRENDS.

DATATYPE operator in a FILTER expression
Support is extended for the SPARQL `DATATYPE` operator in a FILTER expression.

Constants in a FILTER expression
Support is extended for constants in a FILTER expression.

Unary minus in a FILTER expression
A filter expression with a unary minus on variables is not supported.

```
FILTER ( -?v = -10 )
```

The expression returns an RdfStoreException, with error identifier DB255001E and SQL error code -104.

DISTINCT * or REDUCED * operators in a SELECT expression
Support is extended for `DISTINCT *` or `REDUCED *` operators in a SELECT expression.

Data type operator in a SELECT expression
support is extended for the SPARQL data type operator in a SELECT expression.

Dot escape sequence in regular expression
The dot escape sequence in regular expression pattern matches has limitations, where the expression does not properly match the dot character.

```
FILTER regex(?val, "example\\.com")
```

The preceding code sample does not match the string "example.com" as expected.

Double backslash escape sequence limitation
Escape sequences with double backslash in strings do not get interpreted correctly. A workaround is not available.

Cygwin and `createrdfstoreandloader` command (Windows)
When you issue the **`createrdfstoreandloader`** command by using Cygwin on Windows platforms, Cygwin hangs instead of displaying any errors or warning messages. Therefore, issue the **`createrdfstoreandloader`** command only on Linux or UNIX platforms. Then use the generated DB2 load files and SQL files to load to a DB2 server on the Windows platform.

JENA model API support

The DB2 implementation of JENA model API is limited in how you can use the `Model.read()` and `Model.add(Model)` APIs.

Duplicate triples when using the `Model.read()` API
If the input source contains duplicate triples, it is possible that duplicates are not removed, because the JENA library implementation of the `Model.read()` API uses bulk loading in batches of 1000 triples. The DB2 RDF store does not filter duplicate triples across these batches.

As a workaround, always read the input source into an in-memory JENA model and then add the in-memory model to the DB2 store by using the `Model.add(model)` API.

Duplicate triples when using the `Model.add(Model)` API
The `Model.add(Model)` API assumes the graph that is being added does not exist in the data set. If the graph exists and you are adding duplicate triples, the duplicate triple is not removed.

The suggested approach for a DB2 RDF store is listed here:

1. The first time you add a graph, use the `Model.add(Model)` method.
2. If you want to add or update triples to that existing graph, use the following functions:
   - `model.add(s,p,v)`
   - `model.add(statement)`
   - `graph.add(Triple)`

- `Resource.addXX()`

Remember: If you are adding a triple that exists, an error message is returned.

In DB2 Version 10.1 Fix Pack 2 and later fix packs, both of the preceding limitations are now removed in the DB2 product implementation of the JENA API.

Issuing SPARQL queries

You can query data stored in an RDF store.

Example

The following program demonstrates how to query data in an RDF store by using the SPARQL query language.

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

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.rdf.model.Model;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.exception.RdfStoreException;
import com.ibm.rdf.store.jena.RdfStoreFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryExecutionFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryFactory;

public class RDFStoreSampleQuery   {


 public static void main(String[] args) throws SQLException, IOException {

  Connection conn = null;
  Store store = null;
  String storeName = "sample";
  String schema = "db2admin";

  // Get a connection to the DB2 database.
  try {
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   conn = DriverManager.getConnection(
     "jdbc:db2://localhost:50000/dbrdf", "db2admin",
     "db2admin");
  } catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }


  try {

   /* Connect to required RDF store in the specified schema. */
   store = StoreManager.connectStore(conn, schema, storeName);

   /* This is going to be our SPARQL query i.e. select triples
    in the default graph where object is <ibm.com>
    */
      String query = "SELECT * WHERE { ?s ?p
```

```
         <https://www.ibm.com> }";

      /* Create the Query object for the SPARQL string. */
      Query q = RdfStoreQueryFactory.create(query);

      /* Get the Dataset interface of the RDF store. */
      Dataset ds = RdfStoreFactory.connectDataset(store, conn);

      /* Create a QueryExecution object, by providing the query to execute
       and the dataset against which it to be executed. */
      QueryExecution qe = RdfStoreQueryExecutionFactory.create(q, ds);

      long rows = 0;
      Model m = null;

      /* Based on the SPARQL query type, call the proper execution
        method. */
      if (q.isSelectType()) {
       ResultSet rs = qe.execSelect();
       while (rs.hasNext()) {
        QuerySolution qs = rs.next();
        System.out.println(qs);
        System.out.println();
        rows++;
       }
      }
      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");
      }


      /* Close the QueryExecution object. This is required to ensure
       no JDBC statement leaks. */
      qe.close();

      // Display the # of rows returned
      if ( m != null ) {
       System.out.println("Number of Rows  : " + m.size());
       m.close();
      }
      else {
       System.out.println("Number of Rows  : " + rows);
      }

     }
     catch(RdfStoreException e) {

      e.printStackTrace();
     }
     catch(Exception e) {
      e.printStackTrace();
     }

    }

  }
```

Creating a union of all named graphs

You can set the default graph to be the union of all named graphs in the data set for a SPARQL query. This feature applies to queries only. It does not affect the storage nor does it change loading.

The following two programs demonstrate how to set the default graph as the union of all named graphs in the data set for a SPARQL query.

Example

1. The following sample Java program sets the default graph for all queries from a store object:

```
import java.sql.Connection;

import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.Symbols;
import com.ibm.rdf.store.jena.RdfStoreFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryExecutionFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryFactory;


public class UnionDefaultGraph {

 public static void setPerQuery() {

  Connection conn = null;
  Store store = null;

  // get a connection to the DB2 database
  //conn = DriverManager.getConnection(...);

  store = StoreManager.connectStore(conn, "db2admin", "Sample");

  //Set the default graph as the union of all named graphs in the data set
  // for all queries on the store object
  store.getContext().set(Symbols.unionDefaultGraph, true);

  // create the query
  String query = "SELECT * WHERE { ?s ?p <https://www.ibm.com> }";
  Query q = RdfStoreQueryFactory.create(query);
  Dataset ds = RdfStoreFactory.connectDataset(store, conn);
  QueryExecution qe = RdfStoreQueryExecutionFactory.create(q, ds);

  // proceed to execute the query

 }

}
```

2. The following sample Java program sets the default graph on a per query basis:

```
import java.sql.Connection;

import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.Symbols;
import com.ibm.rdf.store.jena.RdfStoreFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryExecutionFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryFactory;


public class UnionDefaultGraph {

 public static void setPerQuery() {

  Connection conn = null;
  Store store = null;

  // get a connection to the DB2 database
  //conn = DriverManager.getConnection(...);


store = StoreManager.connectStore(conn, "db2admin",
"Sample");
  String query = "SELECT * WHERE { ?s ?p <https://www.ibm.com> }";
  Query q = RdfStoreQueryFactory.create(query);
  Dataset ds = RdfStoreFactory.connectDataset(store, conn);
  QueryExecution qe = RdfStoreQueryExecutionFactory.create(q,
ds);

  /* Set the default graph as the union of all named graphs
    * in the data set just for this query.
   */
  qe.getContext().set(Symbols.unionDefaultGraph, true);

  // Proceed to run the query.

 }

}
```

Registering custom DESCRIBE handlers

Use the ARQ defined mechanism to customize how DESCRIBE queries are handled.

A default DESCRIBE handler already comes registered with a DB2 RDF store. It provides a one level depth description of selected resources.

When you implement your own DESCRIBE handlers, ensure that you minimize the number of calls made to the DB2 database server.

Example

The following program demonstrates how to register and implement your own DESCRIBE handler for the DB2 RDF store.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

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.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.rdf.model.AnonId;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.ResourceFactory;
import com.hp.hpl.jena.sparql.core.describe.DescribeHandler;
import com.hp.hpl.jena.sparql.core.describe.DescribeHandlerFactory;
import com.hp.hpl.jena.sparql.core.describe.DescribeHandlerRegistry;
import com.hp.hpl.jena.sparql.util.Context;
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 DescribeTest {

 /**
  * @param args
  * @throws ClassNotFoundException
  * @throws SQLException
  */

 public static void main(String[] args) throws ClassNotFoundException,
   SQLException {
  if (args.length != 5) {
   System.err.print("Invalid arguments.\n");
   printUsage();
   System.exit(0);
  }

    /* Note: ensure that the DB2 default describe handler is also removed.
     * Use the ARQ API's to remove the default registered describe handlers.
     * If you don't do this, every resource runs through multiple describe
     * handlers, causing unnecessarily high overhead.
     */

  /*
   * Now Register a new DescribeHandler (MyDescribeHandler)
   */
  DescribeHandlerRegistry.get().add(new DescribeHandlerFactory() {
   public DescribeHandler create() {
    return new MyDescribeHandler();
   }
  });

  /*
   * Creating database connection and store object.
   */
  Store store = null;
  Connection conn = null;

  Class.forName("com.ibm.db2.jcc.DB2Driver");

  String datasetName = args[0];
  String url = args[1];
  String schema = args[2];
  String username = args[3];
  String passwd = args[4];

  conn = DriverManager.getConnection(url, username, passwd);

  if (StoreManager.checkStoreExists(conn, schema, datasetName)) {
   store = StoreManager.connectStore(conn, schema, datasetName);
  } else {
   store = StoreManager.createStore(conn, schema, datasetName, null);
  }

  /*
   * Creating dataset with test data.
   */
  Dataset ds = RdfStoreFactory.connectDataset(store, conn);

  ds.getDefaultModel().removeAll();
  ds.getDefaultModel().add(getInputData());

  /*
   * Executing a DESCRIBE SPARQL query.
   */
  String sparql = "DESCRIBE <http://example.com/x>";

  Query query = RdfStoreQueryFactory.create(sparql);

  QueryExecution qe = RdfStoreQueryExecutionFactory.create(query, ds);

  Model m = qe.execDescribe();

  m.write(System.out, "N-TRIPLES");

  qe.close();

  conn.close();
 }

 private static void printUsage() {
  System.out.println("Correct usage: ");
  System.out.println("java DescribeTest <DATASET_NAME>");
  System.out.println(" <URL> <SCHEMA> <USERNAME> <PASSWORD>");
 }

 // Creating input data.
 private static Model getInputData() {
  Model input = ModelFactory.createDefaultModel();

  Resource iris[] = {
    ResourceFactory.createResource("http://example.com/w"),
    ResourceFactory.createResource("http://example.com/x"),
    ResourceFactory.createResource("http://example.com/y"),
    ResourceFactory.createResource("http://example.com/z") };

  Property p = ResourceFactory.createProperty("http://example.com/p");
  Node o = Node.createAnon(new AnonId("AnonID"));

  for (int i = 0; i < iris.length - 1; i++) {
   input.add(input.asStatement(Triple.create(iris[i].asNode(), p
     .asNode(), iris[i + 1].asNode())));
  }

  input.add(input.asStatement(Triple.create(iris[iris.length - 1]
    .asNode(), p.asNode(), o)));

  return input;
 }
}

/*
 * Sample implementation of DescribeHandler.
  */
class MyDescribeHandler implements DescribeHandler {

 /*
  * Set to keep track of all unique resource which are
  * required to DESCRIBE.
  */
 private Set <Resource> resources;
      private Model accumulator;
 // Remaining field variables

 public void start(Model accumulator, Context ctx) {
  resources = new HashSet <Resource>();
  this.accumulator = accumulator;
  // Other object declaration as needed.
 }

 public void describe(Resource resource) {
  resources.add(resource);
 }

 public void finish() {
  /*
   * Implement your own describe logic.
     * Add the new triples to 'accumulator' model object.
     * It is best to avoid multiple calls to the database, hence
   * structure your logic accordingly.
     * If you need FullClosure, use the
   * com.ibm.rdf.store.internal.jena.impl.DB2Closure.closure() APIs,
   * instead of com.hp.hpl.jena.sparql.util.Closure.closure().
   */


 }
}
```

Enforcing graph level access control using DB2 database server

You can ensure that a SPARQL query can access only specific RDF graphs by having the DB2 engine enforce the access control.

Procedure

The system predicates metadata table contains the RDF predicates. This metadata is specified to enforce graph level access control during store creation. It also stores the names of the columns on the Direct Primary and Reverse Primary tables which contain the values for these predicates.

Issue the following query:

```
"select * from System_predicates_table>"
```

The sample output:

```
ENTRY_ID   COLNAME        MAPNAME
=======================================================
1          SYSPRED_0      http://myapp.net/xmlns/APPID
1          SYSPRED_1      http://myapp.net/xmlns/CONTEXTID
```

Here `SYSPRED_0` is the column for predicate http://myapp.net/xmlns/APPID and `SYSPRED_1` the column for predicate http://myapp.net/xmlns/CONTEXTID on the Direct Primary and Reverse Primary tables.
You can use the features of DB2 software's fine grained access control to define `ROW LEVEL` constraints as per your requirements on the Direct Primary and Reverse Primary tables using these columns.

Enforcing graph level access control by using the RDF store SQL generator

You can control access to specific RDF graphs. You can control access by having the RDF store SQL generator apply the right filters in the SQL it generates.

Determine the values of RDF predicates based on RDF graphs that contain triples. The values determine which graphs are accessed.

Decide whether the access control check is for a single value or any value in a set of values. If the access control check is for a single value, create a `QueryFilterPredicateEquals` object that returns this value. If the access control check is for any one value in a set of values, create a `QueryFilterPredicateMember` object that returns set of values. Repeat this process for each access control filter predicate.

Set the objects into the `QueryExecution` context for the SPARQL query that is issued.

Example

The following Java program demonstrates how graph access is controlled by using the RDF store SQL generator.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.hp.hpl.jena.query.Dataset;
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.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.ResourceFactory;
import com.ibm.rdf.store.Store;
import com.ibm.rdf.store.StoreManager;
import com.ibm.rdf.store.Symbols;
import com.ibm.rdf.store.jena.RdfStoreFactory;
import com.ibm.rdf.store.jena.RdfStoreQueryExecutionFactory;
import com.ibm.rdf.store.query.filter.QueryFilterPredicate;
import com.ibm.rdf.store.query.filter.QueryFilterPredicateEquals;
import com.ibm.rdf.store.query.filter.QueryFilterPredicateMember;
import com.ibm.rdf.store.query.filter.QueryFilterPredicateProvider;

public class QueryStoreGraphAccessControl {

 /* Property objects for the two RDF predicates based on whose triples
  * access to the graph is controlled.
  */
 private final static Property APPID =
ModelFactory.createDefaultModel()
.createProperty("http://myapp.net/xmlns/APPID");

 private final static Property CONTEXTID =
ModelFactory.createDefaultModel()
   .createProperty("http://myapp.net/xmlns/CONTEXTID");


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

  Connection conn = null;
  Store store = null;
  // get a connection to the DB2 database
  try {
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   conn = DriverManager.getConnection(
     "jdbc:db2://localhost:50000/dbrdf",
"db2admin","db2admin");
  } catch (ClassNotFoundException e1) {
   e1.printStackTrace();
  }

  /* Connect to the access controlled store. */
  store = StoreManager.connectStore(conn, "db2admin",
"SampleAccessControl");
  Dataset ds = RdfStoreFactory.connectDataset(store, conn);

  // Insert some data for querying
  insertDataForQuerying(ds);

  // Query and ensure access control is enforced
  QueryWithGraphAccessControl(ds);

 }


 private static void QueryWithGraphAccessControl(Dataset ds) {

  //Create the filter value for APPID.
  final QueryFilterPredicateEquals appIdFilter =
   new QueryFilterPredicateEquals() {
    public Literal getValue() {
     return ModelFactory.createDefaultModel()
       .createLiteral("App1");
    }
   };

  //Create the filter value set for contextID.
  final QueryFilterPredicateMember ctxIdFilter = new
QueryFilterPredicateMember() {
    public List <> getValues() {
      List<Literal> a = new ArrayList<Literal>();
      a.add(ModelFactory.createDefaultModel()
       .createLiteral("Context1"));
      a.add(ModelFactory.createDefaultModel()
       .createLiteral("Context2"));
      return a;
    }
 };

  // Create the QueryExecution object.
  QueryExecution qe = RdfStoreQueryExecutionFactory.create(
   "select ?who where { ?who <http://pre/test.3> ?x }",
ds);

  // Set the access control filter values for this query.
  qe.getContext().set(Symbols. queryFilterPredicates,
new QueryFilterPredicateProvider() {
    public QueryFilterPredicate getQueryFilterPredicate(
     Property filterProperty) {
    if (filterProperty.equals(APPID) ) {
     return appIdFilter;
    }
    else if ( filterProperty.equals(CONTEXTID)) {
     return ctxIdFilter;
    }
    else
     return null;
   }

   });

  // Set the default graph as a union of all named graphs.
  qe.getContext().set(Symbols.unionDefaultGraph, true);

  /* Execute SPARQL. Note only Model1 will match and Model2
     * triples are not returned */
  ResultSet rs = qe.execSelect();
  while (rs.hasNext()) {
   QuerySolution qs = rs.next();
   System.out.println(qs.toString());
  }
  qe.close();


 }

 private static void insertDataForQuerying(Dataset ds) {

  // Adding triples to graph1.
  ds.getNamedModel("Model1").add(getMemModel());

  // Adding filter predicate triples in the existing graph.
  ds.getNamedModel("Model1").add(
ResourceFactory.createResource(
      "http://res1"), APPID, "App1");

ds.getNamedModel("Model1").add(
ResourceFactory.createResource(
     "http://res1"), CONTEXTID, "Context1");

  // Adding triples to graph2.
  ds.getNamedModel("Model2").add(getMemModel());

  // Adding filter predicate triples in the existing graph.
  ds.getNamedModel("Model2").add(
ResourceFactory.createResource(
      "http://res2"), APPID, "App2");
ds.getNamedModel("Model2").add(
ResourceFactory.createResource(
     "http://res2"), CONTEXTID, "Context2");

 }

 private static Model getMemModel() {private static Model getMemModel() {
  String sub = "http://sub/";
  String pre = "http://pre/test.";
  String obj = "http://obj/";

  Model m = ModelFactory.createDefaultModel();

  long TRIPLE_COUNT = 12;
  for (int i = 0; m.size() < TRIPLE_COUNT; i++) {
   Resource s = ResourceFactory.createResource(sub + (i % 3));
   Property p = ResourceFactory.createProperty(pre + (i % 9));
   Literal o = ResourceFactory.createPlainLiteral(obj + (i % 4));
   m.add(ResourceFactory.createStatement(s, p, o));
  }

  return m;
 }


}
```

Chapter 10. Setting up SPARQL Version 1.1 Graph Store Protocol and SPARQL over HTTP

In DB2 Version 10.1 Fix Pack 2 and later fix packs, DB2 RDF supports the SPARQL Version 1.1 graph store HTTP protocol. This protocol requires Apache JENA Fuseki Version 0.2.4. You need to set up the Fuseki environment to use the SPARQL REST API.

Before you begin

To set up the Fuseki environment:

1. Download the `jena-fuseki-0.2.4-distribution.zip` file from http://archive.apache.org/dist/jena/binaries/"http://archive.apache.org/dist/jena/binaries/".
2. Extract the file on your local system.

Procedure

1. Open a command prompt window and go to the `<`*`Fuseki install dir`*`>/jena-fuseki-0.2.4` directory.
2. Open the `config.ttl` file and add `db2rdf` as a prefix

   ```
   @prefix :        <#> .
   @prefix fuseki:  <http://jena.apache.org/fuseki#> .
   @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
   @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
   @prefix tdb:     <http://jena.hpl.hp.com/2008/tdb#> .
   @prefix ja:      <http://jena.hpl.hp.com/2005/11/Assembler#> .

   @prefix db2rdf:  <http://rdfstore.ibm.com/IM/fuseki/configuration#>
   ```
3. Add the DB2 RDF service to the `config.ttl` file. Add this service to the section of the file where all the other services are registered.

   ```
   fuseki:services (
        <#service1>
        <#service2>
         <#serviceDB2RDF_staffing>
      ) .
   ```

   You can register multiple services. Each service queries different DB2 RDF data sets.
4. Add the following configuration to the `config.ttl` file to initialize the RDF namespace. This configuration registers the assembler that creates the `DB2Dataset`. The configuration also registers the `DB2QueryEngine` and `DB2UpdateEngine` engines.

   ```
   # DB2
   [] ja:loadClass "com.ibm.rdf.store.jena.DB2" .
   db2rdf:DB2Dataset  rdfs:subClassOf  ja:RDFDataset .
   ```
5. Add details about the DB2 RDF service to the end of `config.ttl` file.

   ```
   # Service DB2 Staffing store
   <#serviceDB2RDF_staffing>
   rdf:type fuseki:Service ;
   rdfs:label "SPARQL against DB2 RDF store" ;
   fuseki:name "staffing" ;
   fuseki:serviceQuery "sparql" ;
   fuseki:serviceQuery "query" ;
   fuseki:serviceUpdate "update" ;
   fuseki:serviceUpload "upload" ;
   ```

```
fuseki:serviceReadWriteGraphStore "data" ;
fuseki:serviceReadGraphStore "get" ;
fuseki:serviceReadGraphStore "" ;
fuseki:dataset <#db2_dataset_read> ;
.

<#db2_dataset_read> rdf:type db2rdf:DB2Dataset ;

# specify the RDF store/dataset and schema
db2rdf:store "staffing" ;
db2rdf:schema "db2admin" ;

# Database details. Specify either a jdbcConnectString
# with username and password or specify a jndiDataSource
db2rdf:jdbcConnectString "jdbc:db2://localhost:50000/RDFSAMPL" ;
db2rdf:user "db2admin" ;
db2rdf:password "db2admin" .

#db2rdf:jndiDataSource "jdbc/DB2RDFDS" .
```

6. Issue the following commands from the command line:

```
SET CLASSPATH=./fuseki-server.jar;<DB2_FOLDER>/rdf/lib/rdfstore.jar;
<DB2_FOLDER>/rdf/lib/wala.jar;<DB2_FOLDER>/rdf/lib/antlr-3.3-java.jar;
<DB2_FOLDER>/rdf/lib/commons-logging-1-0-3.jar;<DB2_FOLDER>/java/db2jcc4.jar;
%CLASSPATH%;

java org.apache.jena.fuseki.FusekiCmd --config config.ttl
```

Results

1. Start your browser and load the `localhost:3030` URL. The Fuseki page loads in the browser window.
2. Click the `Control Panel` hyperlink in the browser window and select `Dataset` from the drop-down list. The drop-down list contains all the data sets that are listed in the `config.ttl` file. Select the data set that you configured.
3. Issue a SPARQL query by using the options from the GUI. The first section is for querying the data set with the SPARQL query language. The second section is for modifying the data set with SPARQL updates. The third section is for loading data into the graphs in the data set.

Chapter 11. Maintaining an RDF store

Create either a default or an optimized DB2 RDF store and load RDF data into it. Perform maintenance on your RDF store to achieve optimal performance query performance.

Updating statistics in an RDF store

You can update the statistics for an RDF store.

About this task

To ensure optimal performance query performance in an RDF store, a combination of DB2 database statistics and RDF store specific statistics are maintained for an RDF store.

As part of the RDF store creation process, DB2 **RUNSTATS** command profiles are created for the data tables of the RDF store. By you ensuring that **AUTORUNSTATS** parameter is enabled, DB2 automatically updates distribution statistics on the tables.

In case you have specifically performed operations that have updated significant amounts of data in the RDF store, it is best if you manually invoke DB2 statistics gathering rather than wait for automatic updates. To manually invoke DB2 statistics gathering, invoke the following command for each of the data tables in the RDF store:

```
db2 RUNSTATS ON <table_name> USE PROFILE
```

The data tables are direct_primary, direct_secondary, reverse_primary, reverse_secondary and long_strings tables.

To help ensure that SPARQL queries generate efficient SQL queries, it is important that the most common RDF triples by RDF subject, object and predicate are stored in the Topk_Stats table of the RDF store. To automatically gather information about the most common RDF triples, as part of RDF store creation, a DB2 administrative task is created and registered with the DB2 Administrative Task Scheduler.

However, the interval of this task is not set automatically and therefore will not run until you set it. Use the **schedule** parameter of the **SETSTATSSCHEDULE** command. This parameter accepts a standard CRON format string to represent the interval at which statistics gathering should be invoked. Generally, you don't need to set the frequency of this task to anything smaller than an hour.

Procedure

- Use the following command to set the statistics gathering schedule to every 1 hour on the hour for store rdfStore2 on database DB1.
  ```
  setstatsschedule rdfStore2 -host localhost -port 60000
  -db DB1 -user db2admin -password XXX
  -schema db2admin -schedule "*/59 * * * *"
  ```
- In case you have specifically performed operations that have updated significant amounts of data in the RDF store, manually gather these statistics instead of waiting for the scheduler. To manually invoke this run the **updaterdfstorestats** command.

For example, use the following command to update statistics for store rdfstore2 on database DB1 and schema db2admin for host localhost and port 60000:

```
updaterdfstorestats rdfstore2 -host localhost -port 60000 -db DB1
-user db2admin -password XXX
-schema db2admin
```

Converting a default store to an optimized RDF store

If you started with a default RDF store, you can use the **reorgcheckrdfstore** command to verify if the store should be optimized. You can also do this for an optimized store in which the predicate correlation has significantly changed. Then use the **reorgrdfstore** and **reorgswitchrdfstore** commands to move to an optimized RDF store.

First, verify if an RDF store needs reorganizing. Then create reorganized tables for an RDF store. Finally, switch to the reorganized tables.

Verifying if an RDF store needs reorganizing

If the predicate correlation of the RDF data has changed and significant amounts of data has been inserted with this change, the number and length of columns for tables in the store as well as assignment of predicates to columns might no longer be optimal for your data.

If these values are no longer optimal, query and insert performance for the RDF store might be negatively affected.

Procedure

To determine whether a store requires reorganization, issue the **reorgcheckrdfstore** command. For example, for a store myRdfStore in database DB1, issue the following command:

```
reorgcheckrdfstore myRdfStore -db DB1
-user db2admin -password db2admin
```

If the tables do not require reorganization, the following message is displayed:

```
No reorganization is required for store myRdfStore.
```

If any tables require reorganization, the output lists the tables, as shown in the following example:

```
TABLENAME                 REORG    ColsRequired    ColSize
-----------------------------------------------------------
direct_primary_hash        true               5       118
reverse_primary_hash       true               3       118
-----------------------------------------------------------
```

What to do next

Use the **reorgrdfstore** command to create reorganized tables. For details, see topic about creating reorganized tables for an RDF store.

Creating reorganized tables for an RDF store

If the **reorgcheckrdfstore** command indicates that you must reorganize tables, use the **reorgrdfstore** command to create and populate reorganized tables.

Before you begin

Ensure that no updates are made to the RDF store while the **reorgrdfstore** command is in progress by changing all the tables of the RDF store to read-only mode.

Procedure

To reorganize tables in an RDF store, issue the **reorgrdfstore** command. For example, for store myRdfStore in database DB1, issue the following command to create a new reorganized table for table direct_primary_hash:

```
reorgrdfstore myRdfStore -db DB1
-user db2admin -password db2admin
-table direct_primary_hash -newtablenamesprefix reorgd
```

The name of the new table is reorgd_*original_table_name* because you specified reorgd as the prefix for the new table name for the command. The *original_table_name* value represents the name that was set for the direct_primary_hash table in the `objectNames.props` properties file.

Results

The time to reorganize tables depends on the amount of data in the store.

If no reorganization is required for a table that you specify for the **table** parameter, a message indicates that.

What to do next

Change the store to use the new reorganized tables. For details, see topic about switching to reorganized tables in an RDF store.

Switching to reorganized tables in an RDF store

The **reorgrdfstore** command creates reorganized tables, but the RDF store does not use those new tables until you issue the **reorgswitchrdfstore** command.

Before you begin

Ensure that all clients of the RDF store are disconnected.

Procedure

To update an RDF store with reorganized tables, issue the **reorgswitchrdfstore** command. For example, for store myRdfStore in database DB1, issue the command as follows:

```
reorgswitchrdfstore myRdfStore -db DB1
-user db2admin -password db2admin
```

Results

The **reorgswitchrdfstore** command renames the original tables to old_*original_table_name*, and the reorganized tables use the original names.

When the RDF store is switched to a reorganized store, the tables are not renamed. The store will be modified to start using the new tables, and the old tables will remain as is.

What to do next

Reconnect clients to the store.

Drop the old tables if required.

Chapter 12. RDF commands

RDF commands provide extensive user control, customization, and personalization. You can use these commands to perform a variety of store creation, administration and query related tasks.

createrdfstore command

The **createrdfstore** command creates an empty RDF store without any data.

To create an optimized RDF store that is uses existing data, use the **createrdfstoreandloader** command instead.

Command syntax

```
►►──createrdfstore──storeName──┬─────────────────────────┬──┬───────────────────┬──────►
                               └─-objectnames──objNames──┘  └─-host──hostName───┘

►──┬─────────────────────┬──-db──dbName──-user──userName──-password──password──────────►
   └─-port──portNumber───┘

►──┬─────────────────────────┬──┬───────────────────────────────────────────────┬─────►
   └─-schema──schemaName─────┘  └─-predicatemappings──predicateMappingsFileName─┘

►──┬───────────────────────────────────────────────────┬──────────────────────────────►◄
   └─-systempredicates──systemPredicatesFileName───────┘
```

Command parameters

-storename *storeName*
: Specifies a name for the RDF store. Ensure that the name satisfies the rules for DB2 database table names.

-objectnames *objNames*
: Specifies a Java properties file that lists the names of the RDF store tables. This file can be any valid file name, but it must have the extension ".properties".

  If you do not specify the **objectNames** parameter, system generated table names are used instead.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
: Specifies the port number of the database.

-db *dbName*
: Specifies the database to which a connection is established. The minimum database page size is 32 KB.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
: Specifies the password that is used to establish the connection.

-schema *schemaName*
: Specifies the database schema in which to create the RDF store.

-predicatemappings *predicateMappingsFileName*
: In DB2 Version 10.1 Fix Pack 2 and later fix packs, specifies the path of the file that contains the predicate mappings that are to be used in the store. The mappings occur between the predicates and their assigned columns. The mappings are computed based on predicate occurrence.

-systempredicates *systemPredicatesFileName*
: In DB2 Version 10.1 Fix Pack 2 and later fix packs, specifies the properties file that contains the filter predicates that are to be applied to the query. These system predicates are stored in an RDF store to help enable graph level access control.

Example

Example 1: The following command creates a store named rdfStore1 in database DB1 with port 60000 and schema db2admin on host localhost:

```
createrdfstore rdfStore1 -host localhost -port 60000 -db DB1
-user db2admin -password XXX -schema db2admin
```

Example 2: The following command creates a store named rdfstore2 in database DB1 with port 60000 and schema db2admin by using system predicates from the syspreds.props file and predicate mappings from the `predicatemappings.nq` file.

```
createrdfstore rdfStore1 -host localhost -port 60000 -db DB1
-user db2admin -password XXX -schema db2admin
-predicatemappings predicatemappings.nq -systempredicates syspreds.props
```

Usage notes

You must issue command and parameter names in lowercase.

createrdfstoreandloader command

The **createrdfstoreandloader** command analyzes RDF data and creates an empty RDF store whose schema is optimized for the existing RDF data. This command also generates the load files and the commands to load the store from these loader files.

Command syntax

```
>>--createrdfstoreandloader--storeName--+-------------------------+------------>
                                        '--objectnames--objNames--'

>--+--------------------+--+---------------------+--db--dbName---user--userName------>
   '--host--hostName--'    '--port--portNumber--'

>--password--password--+---------------------+--rdfdata--rdfDataFile------------>
                       '--schema--schemaName--'

>--+------------------------------+--+---------------------------+-------------->
   '--storeloadfile--loadCommandFile--'  '--storeschemafile--ddlFile--'

>--+------------------------------------------------+---------------------------><
   '--systempredicates--systemPredicatesFileName--'
```

Command parameters

-storename *storeName*
: Specifies a name for the RDF store. Ensure that the name satisfies the rules for DB2 database server table names and is unique within the database schema.

-objectnames *objNames*
: Specifies a Java properties file that lists the names of the RDF store tables. This file can be any valid file name, but it must have the extension ".properties".

  If you do not specify the **objectNames** parameter, system generated table names are used instead.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
: Specifies the port number of the database.

-db *dbName*
: Specifies the database to which a connection is established. The minimum database page size is 32 KB.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
: Specifies the password that is used to establish the connection.

-schema *schemaName*
: Specifies the database schema in which to create the RDF store.

-rdfdata *rdfDataFile*
: Specifies a file with the RDF data from which the optimized RDF store schema is created. Also, load files are created based on the RDF data in this file.

-storeloadfile *loadCommandFile*
: Specifies the output path of the file with the commands to load data into the RDF store. You can run this file by using the DB2 command line processor (CLP).

  If you do not specify the **-storeloadfile** parameter, a file with the name `loadCommands.sql` is created in the current folder.

  The load files are created in the folder where the file with the commands is created.

-storeschemafile *ddlFile*
: Specifies the output path and file where the DDL scripts are generated for the store.

  If you do not specify the **-storeschemafile** parameter, the DDL scripts file is not created.

-systempredicates *systemPredicatesFileName*
: In DB2 Version 10.1 Fix Pack 2 and later fix packs, specifies the properties file that contains the filter predicates to be applied to the query. These system predicates are stored in a DB2 RDF store to help enable graph level access control.

Example

The following command creates a store named rdfStore1 in database DB1 with port 60000 and schema db2admin on host localhost. The input RDF data file is `myRdfData.nq`, and the name of the generated store loader file is `load.sql` in the `./rdfLoader/` directory.

```
createrdfstoreandloader rdfStore1 -host localhost -port 60000 -db DB1
-user db2admin -password XXX -schema db2admin
-rdfdatafile ./myRdfData.nq -storeloadfile ./rdfLoader/load.sql
```

Usage notes

You must issue the command and parameter names in lowercase.

The directory from which the command is issued must not have a space in its path. The **`storeloadfile`** and **`storeschemafile`** parameters must not have a space in their path either. On Windows platforms, if any paths that are specified contain a space in the folder or file name, the whole string must be enclosed within double quotation marks.

On Windows platforms, the **`createrdfStoreAndLoader`** command requires the CygWin application. The Gawk utility required for this command is Versions 4.0 or later. The Core utility required for this command is Version 8.14 or later. After installing CygWin, add `<`*`CgyWin_install_directory`*`>/bin` to the PATH environment variable. If you do not have CygWin on the path, the following error message is displayed when you run the command:

```
'Cannot run program "sh": CreateProcess error=2, The system cannot find
the specified file.'
```

On Windows platforms, you can start the **`createrdfStoreAndLoader`** command either from a CygWin command prompt or default command prompt. When using a CygWin command prompt, all file paths (-rdfdata, -storeloadfile, -storeschemafile, -objectnames) must not include the 'cygdrive' prefix. Instead use a Windows path such as `C:\....`.

droprdfstore command

The **`droprdfstore`** command removes an existing RDF store.

Command syntax

```
►►──droprdfstore──storename──┬────────────────────┬──┬────────────────────┬──────►
                             └─-host──hostName─┘  └─-port──portNumber─┘

►──┬───────────────┬──┬───────────────────┬──┬───────────────────────┬──────────►
   └─-db──dbName─┘  └─-user──userName─┘  └─-password──password─┘

►──┬───────────────────────┬────────────────────────────────────────────────────►◄
   └─-schema──schemaName─┘
```

Command parameters

-storename
: Specifies a name identifying the tripleStore within a database or schema.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
Specifies the port number of the database.

-db *dbName*
Specifies the database to which a connection is established.

-user *userName*
Specifies the authorization name that is used to establish the connection,

-password *password*
Specifies the password that is used to establish the connection.

-schema *schemaName*
Specifies the database schema in which the RDF is located.

Example

Issue the **droprdfstore** command to remove a store named rdfStore4 in database DB1 and schema db2admin for host localhost and port 60000.

```
droprdfstore rdfStore4 -host localhost -port 60000 -db DB1
-user db2admin -password XXX
-schema db2admin
```

Usage notes

- Command and parameter names must be issued in lowercase.

genpredicatemappings command

In DB2 Version 10.1 Fix Pack 2 and later fix packs, the **genpredicatemappings** command generates predicate mappings based on predicate correlation for an RDF store.

Command syntax

```
►►──genpredicatemappings──storeName──┬────────────────┬──────────────────────►
                                     └─-host──hostName─┘

►──┬────────────────────┬──-db──dbName──-user──userName──-password──password──────►
   └─-port──portNumber──┘

►──-schema──schema──outputFile──────────────────────────────────────────────►◄
```

Command parameters

storeName
Specifies the RDF store.

-host *hostName*
Specifies the host where the database is located.

-port *portNumber*
Specifies the port number of the database.

-db *dbName*
Specifies the database to which a connection is established.

-user *userName*
Specifies the authorization name that is used to establish the connection.

-password *password*
: Specifies the password that is used to establish the connection.

-schema *schemaName*
: Specifies the database schema for the RDF store.

outputFile
: Specifies the path and name of the file to which the mappings are written. If an output file is not specified, the output is written to the console.

Example

The following command generates predicate mappings for an RDF store named MyStore and writes the output to a file:

```
genpredicatemappings MyStore -db RDFSAMPL -user db2admin
-password db2admin "C:\MyStore_predicate_mappings.txt"
```

Usage notes

You must issue the command and parameter names in lowercase.

loadrdfstore command

In DB2 Version 10.1 Fix Pack 2 and later fix packs, the **loadrdfstore** command loads triples to an existing RDF store.

Command syntax

```
►►──loadrdfstore──storeName──┬────────────────┬──┬────────────────────┬──────────►
                             └─-host──hostName─┘  └─-port──portNumber─┘

►──-db──dbName───-user──userName───-password──password──┬──────────────────┬──────►
                                                         └─-schema──schema─┘

►──┬──────────┬──────────────────────────────────────────────────────────────────►◄
   └─loadFile─┘
```

Command parameters

storeName
: Specifies the RDF store to be queried.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
: Specifies the port number of the database.

-db *dbName*
: Specifies the database to which a connection is established.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
: Specifies the password that is used to establish the connection.

-schema *schemaName*
: Specifies the database schema for the RDF store.

loadFile
: Specifies the file that contains the triples to be loaded. The file can be of type nquad, ntriple, or rdfxml. Ntriple and rdfxml files are loaded to the default graph only.
- The extension for nquad files is .nq.
- The extension for ntriples files is .nt.
- The extension for rdfxml files is .rdf or .xml.

Example

The following command loads a file that contains triples for an RDF store named myStore in the RDFSAMPL database.

```
loadrdfstore myStore -db RDFSAMPL -user db2admin
-password db2admin -schema db2admin c:\simple.nq
```

Usage notes

You must issue the command and parameter names in lowercase.

queryrdfstore command

In DB2 Version 10.1 Fix Pack 2 and later fix packs, you can use the **queryrdfstore** command to query an RDF store from the command line. You can run this query from a file or by specifying it inline as an argument to the **queryrdfstore** command.

Command syntax

```
>>--queryrdfstore--storeName--+--------------------+--+-----------------------+-------->
                              '--host--hostName----'  '--port--portNumber-----'

>---db--dbName---user--userName---password--password--+--------------------+---------->
                                                      '--schema--schema----'

>--+-------------------------------------------------+--+--------------------+-------->
   '--uniondefaultgraph--unionDefaultGraphValue------'  '--reopt--reoptValue--'

>--+--------------------+--+--------------------+-------------------------------------><
   '--file--queryFile---'  '-queryStringValue---'
```

Command parameters

storeName
: Specifies the RDF store to be queried.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
: Specifies the port number of the database.

-db *dbName*
: Specifies the database to which a connection is established.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
  Specifies the password that is used to establish the connection.

-schema *schemaName*
  Specifies the database schema for the RDF store.

-uniondefaultgraph*unionDefaultGraphValue*
  Specifies whether a union default graph is being used. The value can be `true` or `false`.

-reopt*reoptValue*
  Specifies the repeat options for running the query. The options are `once`, `always`, or `none`. The default value is `none`.

-file*queryFile*
  Specifies the file that contains the SPARQL query.

queryStringValue
  Specifies the SPARQL query issued as a string.

Example

Example 1: The following command specifies a query for triples in an RDF store named myStore and in the RDFSAMPL database on a local system, with the **unionDefaultGraph** parameter set to `true`You can query all the triples in an RDF store name myStore in the RDFSAMPL database on the local system, with the **unionDefaultGraph** parameter set to `true`.

```
queryrdfstore myStore -db RDFSAMPL -user db2admin
-password db2admin -schema db2admin
-uniondefaultgraph true "select * where {?s ?p ?v}"
```

Example 2: The following command specifies query usage within in a text file.

```
queryrdfstore myStore -db RDFSAMPL -user db2admin
-password db2admin -schema db2admin
-uniondefaultgraph true -file "C:\query.txt"
```

Usage notes

You must issue the command and parameter names in lowercase.

You can specify the query either within a file or as a parameter for the command, but not both.

reorgcheckrdfstore command

The **reorgcheckrdfstore** command checks whether you must reorganize an RDF store.

The **reorgcheckrdfstore** command identifies whether the number of columns or the column lengths in one or more RDF store tables are not optimal. After running the **reorgcheckrdfstore** command, issue the **reorgrdfstore** command to reorganize the identified tables to help improve query performance.

Command syntax

```
▶▶──reorgcheckrdfstore──-storename──┬────────────────────┬──┬──────────────────────┬──────────▶
                                    └─-host──hostName─┘  └─-port──portNumber─┘
```

```
►─-db─dbName──-user─userName──-password─password─┬──────────────────┬─►◄
                                                  └─-schema─schemaName─┘
```

Command parameters

-storename
"Specifies the name of a tripleStore within a database or schema.

-host *hostName*
Specifies the host where the database is located.

-port *portNumber*
Specifies the port number of the database.

-db *dbName*
Specifies the database to which a connection is established.

-user *userName*
Specifies the authorization name that is used to establish the connection.

-password *password*
Specifies the password that is used to establish the connection.

-schema *schemaName*
Specifies the database schema in which the RDF store is located.

Example

The following command checks whether a store named rdfStore3 must be reorganized. The store is in database DB1 with port 60000 and schema db2admin on host localhost.

```
reorgcheckrdfstore rdfStore3 -host localhost -port 60000 -db DB1
-user db2admin -password XXX
-schema db2admin
```

Only tables requiring reorganization are listed in the output of the **reorgcheckrdfstore** command. The output contains the following columns of data:

tablename
Logical name of the RDF store table.

reorg A true or false value indicating whether the table requires reorganization.

colsrequired
Required number of columns.

colsize
Optimal column length.

Usage notes

You must issue the command and parameter names in lowercase.

reorgrdfstore command

The **reorgrdfstore** command creates new reorganized tables with the optimal number and length of columns for an RDF store based on the existing data in the store. The command also optionally generates files with the new tables' DDL and the load commands to load data into the reorganized tables.

This command can take some time to complete based on the amount of data that needs to be unloaded, creation of load files, and loading the data into reorganized tables. While the command is running, the RDF store data tables should be made read-only.

Command syntax

```
►►──reorgrdfstore──-storename──┬──────────────────┬──┬──────────────────────┬────────►
                               └─-host──hostName──┘  └─-port──portNumber────┘

►─-db──dbName──-user──userName──-password──password──┬──────────────────────────┬───►
                                                     └─-schema──schemaName──────┘

►─-table────tableNameList──-newtablenamesprefix──prefix─────────────────────────────►

►─┬──────────────────────────────────────┬──────────────────────────────────────────►◄
  └─-tablesloadfile──loadCommandFile─────┘
```

Command parameters

-storename
Specifies the name of the tripleStore within a database or schema.

-host *hostName*
Specifies the host where the database is located.

-port *portNumber*
Specifies the port number of the database.

-db *dbName*
Specifies the database to which a connection is established.

-user *userName*
Specifies the authorization name that is used to establish the connection.

-password *password*
Specifies the password that is used to establish the connection.

-schema *schemaName*
Specifies the database schema in which the RDF store is located.

-table *tableNameList*
Specifies the list of logical names of the tables to reorganize. Use the pipe character to separate multiple table names. The list must be contained within double quotation marks.

-newtablenamesprefix *prefix*
Specifies a prefix for the new tables.

-tablesloadfile *loadCommandFile*
Specifies the output path for the file containing the new tables' DDL and commands to load data into the reorganized tables.

If you do not specify the **-tablesloadfile** parameter, a file with the name `loadCommands.sql` is created in the current folder.

The load files are created in the folder where the load command file is created. These files can be run using a CLP window.

Example

The following command reorganizes the tables for a store named rdfStore3 in database DB1 with port 60000 and schema db2admin on host localhost for tables direct_primary_hash and reverse_primary_hash using newtablenames prefixed with reorg:

```
reorgrdfstore rdfStore3 -host localhost -port 60000
-db DB1 -user db2admin -password XXX
-schema db2admin -table "direct_primary_hash|reverse_primary_hash"
-newtablenamesprefix reorg
```

Usage notes

- You mustissue the **reorgrdfstore** command on the machine where the DB2 database server and the RDF store are located.
- You must issue the command and parameter names in lowercase.
- After the successful completion of this command, tables will be automatically loaded. This file can be run using a CLP window.
- After this command completes, the store will continue to use the old tables and must be switched to start using the new tables. To use the reorganized tables issue the **reorgswitchrdfstore** command.

reorgswitchrdfstore command

The **reorgswitchrdfstore** command switches a RDF Store to use newly reorganized tables that were created using the **reorgrdfstore** command.

All clients of this RDF Store should disconnect before this operation and reconnect back after the command is complete.

The completion time does not depend on the amount of data in the store.

Command syntax

```
►►──reorgswitchrdfstore──storename──┬────────────────────┬──┬──────────────────────┬──────►
                                    └─-host──hostName─┘  └─-port──portNumber─┘

►──-db──dbName──-user──userName──-password──password──┬─────────────────────────┬─────►◄
                                                      └─-schema──schemaName─┘
```

Command parameters

-storename
: Specifies a name identifying the tripleStore within a database or schema.

-host *hostNames*
: Specifies the host where a store is created.

-port *portNumber*
: Specifies the port number.

-db *dbName*
: Specifies the database to which a connection is established.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
Specifies the password that is used to establish the connection.

-schema *schemaName*
Specifies the database schema in which the RDF store is located.

Example

Issue the reorgswitchrdfstore command to switch to reorg'd tables for the store named rdfStore3 in database DB1 and schema db2admin for host localhost and port 60000.

```
reorgswitchrdfstore rdfStore3 -host localhost -port 60000 -db DB1
-user db2admin -password XXX
-schema db2admin
```

Usage notes

- Command and parameter names must be issued in lowercase.

setstatsschedule command

The **setstatsschedule** command schedules the automatic update of RDF store statistics.

Command syntax

```
►►──reorgrdfstore──-storename──┬──────────────────┬──┬──────────────────────┬────────►
                               └─-host──hostName─┘  └─-port──portNumber─┘

►──┬──────────────┬──┬──────────────────┬──┬──────────────────────┬─────────────────►
   └─-db──dbName─┘  └─-user──userName─┘  └─-password──password─┘

►──┬──────────────────────┬─-schedule──schedule──────────────────────────────────►◄
   └─-schema──schemaName─┘
```

Command parameters

-storename
Specifies the name of the tripleStore within a database or schema.

-host *hostName*
Specifies the host where the database is located.

-port *portNumber*
Specifies the port number of the database.

-db *dbName*
Specifies the database to which a connection is established.

-user *userName*
Specifies the authorization name that is used to establish the connection.

-password *password*
Specifies the password that is used to establish the connection.

-schema *schemaName*
Specifies the database schema in which the RDF store is located.

-schedule *schedule*
Specifies the schedule for statistics updates in UNIX CRON format. This parameter must be specified within double quotation marks.

Example

The following command schedules the auto updates of statistics to run at 15th minute of every hour for a store named RDFStore in database RDFDB, port 60000 and schema db2admin on host localhost:

```
setStatsSchedule RDFStore -host localhost -port 60000
-db RDFDB -user db2admin -password XXX
-schema db2admin -schedule "15 * * * *"
```

Usage notes

- You must issue the command and parameter names in lowercase.

updaterdfstorestats command

The **updaterdfstorestats** command updates the statistics to reflect current data in an RDF store.

Command syntax

```
►►──updaterdfstorestats──storeName──┬────────────────┬──┬────────────────────┬──►
                                    └─-host──hostName─┘  └─-port──portNumber─┘

►──┬──────────────┬──┬──────────────────┬──┬────────────────────────┬──────────►
   └─-db──dbName─┘  └─-user──userName─┘  └─-password──password─┘

►──┬──────────────────────┬─────────────────────────────────────────────────►◄
   └─-schema──schemaName─┘
```

Command parameters

-storename
: Specifies a name for the store. The name must be unique within a database or schema.

-host *hostNames*
: Specifies the host where the database is located.

-port *portNumber*
: Specifies the port number of the database.

-db *dbName*
: Specifies the database to which a connection is established.

-user *userName*
: Specifies the authorization name that is used to establish the connection.

-password *password*
: Specifies the password that is used to establish the connection.

-schema *schemaName*
: Specifies the database schema in which the RDF store is located.

Example

The following command updates statistics for a store named rdfStore3 in database DB1 with port 60000 and schema db2admin on host localhost:

```
updaterdfstorestats rdfStore3 -host localhost -port 60000 -db DB1
-user db2admin -password XXX
-schema db2admin
```

Usage notes

- You must issue the command and parameter names in lowercase.

Part 2. Appendixes

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command-line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 3. DB2 technical information

Name	Form Number	Available in print	Availability date
Administrative API Reference	SC27-5506-00	Yes	July 28, 2013
Administrative Routines and Views	SC27-5507-00	No	July 28, 2013
Call Level Interface Guide and Reference Volume 1	SC27-5511-00	Yes	July 28, 2013
Call Level Interface Guide and Reference Volume 2	SC27-5512-00	Yes	July 28, 2013
Command Reference	SC27-5508-00	Yes	July 28, 2013
Database Administration Concepts and Configuration Reference	SC27-4546-00	Yes	July 28, 2013
Data Movement Utilities Guide and Reference	SC27-5528-00	Yes	July 28, 2013
Database Monitoring Guide and Reference	SC27-4547-00	Yes	July 28, 2013
Data Recovery and High Availability Guide and Reference	SC27-5529-00	Yes	July 28, 2013
Database Security Guide	SC27-5530-00	Yes	July 28, 2013
DB2 Workload Management Guide and Reference	SC27-5520-00	Yes	July 28, 2013
Developing ADO.NET and OLE DB Applications	SC27-4549-00	Yes	July 28, 2013
Developing Embedded SQL Applications	SC27-4550-00	Yes	July 28, 2013
Developing Java Applications	SC27-5503-00	Yes	July 28, 2013
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-5504-00	No	July 28, 2013
Developing RDF Applications for IBM Data Servers	SC27-5505-00	Yes	July 28, 2013
Developing User-defined Routines (SQL and External)	SC27-5501-00	Yes	July 28, 2013
Getting Started with Database Application Development	GI13-2084-00	Yes	July 28, 2013

Table 3. DB2 technical information (continued)

Name	Form Number	Available in print	Availability date
Getting Started with DB2 Installation and Administration on Linux and Windows	GI13-2085-00	Yes	July 28, 2013
Globalization Guide	SC27-5531-00	Yes	July 28, 2013
Installing DB2 Servers	GC27-5514-00	Yes	July 28, 2013
Installing IBM Data Server Clients	GC27-5515-00	No	July 28, 2013
Message Reference Volume 1	SC27-5523-00	No	July 28, 2013
Message Reference Volume 2	SC27-5524-00	No	July 28, 2013
Net Search Extender Administration and User's Guide	SC27-5526-00	No	July 28, 2013
Partitioning and Clustering Guide	SC27-5532-00	Yes	July 28, 2013
pureXML Guide	SC27-5521-00	Yes	July 28, 2013
Spatial Extender User's Guide and Reference	SC27-5525-00	No	July 28, 2013
SQL Procedural Languages: Application Enablement and Support	SC27-5502-00	Yes	July 28, 2013
SQL Reference Volume 1	SC27-5509-00	Yes	July 28, 2013
SQL Reference Volume 2	SC27-5510-00	Yes	July 28, 2013
Text Search Guide	SC27-5527-00	Yes	July 28, 2013
Troubleshooting and Tuning Database Performance	SC27-4548-00	Yes	July 28, 2013
Upgrading to DB2 Version 10.5	SC27-5513-00	Yes	July 28, 2013
What's New for DB2 Version 10.5	SC27-5519-00	Yes	July 28, 2013
XQuery Reference	SC27-5522-00	No	July 28, 2013

Table 4. DB2 Connect-specific technical information

Name	Form Number	Available in print	Availability date
DB2 Connect Installing and Configuring DB2 Connect Personal Edition	SC27-5516-00	Yes	July 28, 2013
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-5517-00	Yes	July 28, 2013
DB2 Connect User's Guide	SC27-5518-00	Yes	July 28, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.
For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com®.

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is http://pic.dhe.ibm.com/infocenter/db2luw/v10r1.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is http://pic.dhe.ibm.com/infocenter/db2luw/v9r8/.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

Trademarks

Index

C

commands
- RDF
  - createrdfstore 47
  - createrdfstoreandloader 48
  - genPredicateMappings 51
  - loadrdfstore 52
  - queryrdfstore 53
  - reorgcheckrdfstore 54
  - reorgrdfstore 56
  - updaterdfstorestats 59

createrdfstore command 47

D

DB2 Information Center
- versions 66

documentation
- overview 63
- PDF files 63
- printed 63
- terms and conditions of use 66

droprdfstore command 50

G

genpredicatemappings command 51

H

help
- SQL statements 66

J

JENA model API support 30

L

loadrdfstore command 52

N

notices 69

Q

queryrdfstore command 53

R

RDF
- access control for RDF stores 7
- administrative database objects 5
- APIs 26, 29, 30
- checking whether to reorganize RDF store 44

RDF *(continued)*
- commands
  - createrdfstore 47
  - createrdfstoreandloader 48
  - droprdfstore 50
  - genpredicatemappings 51
  - loadrdfstore 52
  - overview 47
  - queryrdfstore 53
  - reorgcheckrdfstore 54
  - reorgrdfstore 56
  - reorgswitchrdfstore 57
  - setstatsschedule command 58
  - updaterdfstorestats 59
- converting default store to optimized RDF store 44
- creating RDF stores
  - default store 15
  - optimized stores 16, 18
  - overview 15
- DB2 Version 9.7 14
- default RDF store 9
- downloads and resources 3
- enforcing graph-level access control
  - DB2 database server 37
  - RDF store SQL generator 38
- environment 13
- graphs
  - creating 26
  - creating union of all named graphs 33
  - deleting 26
  - updating 25, 26
- JENA APIs 30
- maintaining RDF stores 43
- modifying RDF stores
  - overview 23
  - SPARQL UPDATE APIs 26
- optimized RDF stores
  - creating 16
  - overview 9
- overview 1
- queries
  - issuing SPARQL queries 31
  - overview 29
  - restrictions 29
- RDF store tables
  - overview 5
  - reorganized 45
  - reorganizing 45
- RDF stores that use graph-level access control
  - creating 20
- registering custom DESCRIBE handlers 34
- SPARQL 1.1 graph store HTTP protocol 41
- SPARQL UPDATE support 25
- stores
  - modifying data 23
- updating an RDF store 23
- updating RDF stores
  - overview 23
  - SPARQL UPDATE APIs 26
  - statistics 43
- viewing RDF stores 11

reorgcheckrdfstore command 54
reorgrdfstore command 56
reorgswitchrdfstore command 57
Resource Description Framework
  See RDF 1

S

SQL statements
  help
    displaying 66

T

terms and conditions
  publications 66

U

updaterdfstorestats command 59

Printed in USA

SC27-5505-00

Spine information:

IBM DB2 10.5 for Linux, UNIX, and Windows

Developing RDF Applications for IBM Data Servers