Using the DAWG Test Cases with Relational Databases

Matthew Gheen
BBN Technologies
Arlington, VA 22209
mgheen@bbn.com

Introduction

As the Semantic Web has matured, it has become it has become clear that there is a need to integrate older technologies. One such technology is relational databases. There have been a few attempts to make relational databases semantically searchable using SPARQL. These include Free University of Berlin's D2RQ[1] and BBN's Semantic Bridge for Relational Databases (SBRD)[2]. Both allow SPARQL queries to be posed to a relational database, but how do they hold up against the W3C RDF Data Access Working Group (DAWG) test cases[3]? As this question was evaluated, a new question arose: How well do the DAWG test cases apply to these tools? The difficulties experienced in converting the DAWG test cases for these tools show that there is a need to create a specialized subset of test cases. The results in this paper were based upon tests conducted using SBRD and the first revision of the DAWG test cases.

Converting the Test Cases

For this investigation, a program was created to automatically convert the test data in the test cases into SQL. This includes the table `CREATE` and `INSERT` statements required to instantiate the data in a relational database. This translation is straightforward. The objects of rdf:type statements translate to tables, and predicates of all other statements translate to columns. Statements that do not have a subject that is shared with a typed statement are considered untyped. For these statements a table called Anon was used. Similarly, if the query asked for untyped values, the type was assumed to be Anon. Also, *mailto* URI's were converted to strings and stored in the database to prevent the creation of foreign key constraints to non-existent tables. This problem is discussed later in the paper. As these SQL files were generated, a mapping data was simultaneously created that defined how RDF data types were mapped to the database. This ontology was used in the translation from SPARQL to SQL and from the SQL result set back to RDF. See Appendix A for an example of converted RDF to SQL.

To run these newly converted DAWG tests, the data needed to be loaded into a database. First, a simple program was written to load an instance of Apache's Derby database. Next, the table creation and data insertion script was executed. Then, an instance of SBRD was started, and the query was run. The result model was then compared with the expected results; if the model matched the expected result set, the test was considered a success.

The first step in this testing process eliminated a majority of the DAWG test cases from being useful in a relational database environment. Many of the provided datasets

[1] http://sites.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/
[2] http://asio.bbn.com/sbrd.html
[3] http://www.w3.org/2001/sw/DataAccess/tests/

contained data that was not relational database compatible. RDF provides greater flexibility in how data is modeled. It provides a multidimensional abstraction of the data that flat tables in a relational database just cannot capture. Since the test data is provided in RDF format, conversion to a relational database format can be troublesome. The data conversion program revealed four types of incompatible data.

The first incompatibility problem occurred because RDF allows multiple data types for a single predicate. For instance:

```
:x1 :p  “string” .
:x2 :p  “1234”^^xsd:integer .
```

This would translate to storing different data types in a typed column of a database. This problem could be circumvented by storing the data as a *blob*; however, the type granularity would be lost when executing queries.

The second incompatibility problem occurred because of untyped statements. In sort-3’s data set[4], there are four instances of data. The first three are typed as *foaf:Person*; however, the fourth is not explicitly typed. This is an excerpt of the data:

```
_:c rdf:type foaf:Person ;
    foaf:mbox <mailto:fred@work.example> ;
    foaf:name "Fred" .

_:e foaf:name "Bob" .
```

The source of the last statement cannot be determined since there is no explicit typing. Therefore, it is stored in a separate table in the database, named Anon. This causes the test query for all *foaf:name*s to return incorrect results. If the query is typed as Person, three names would be returned, since foaf:name is mapped to the Person table. If the query is untyped, the Anon type is assumed, and only one name would be returned.

The third incompatibility problem occurs where there is more than one value for a subject-predicate pair. This is the relational database equivalent of having two values for the same column and row. This type of multivalued data is helpful when modeling objects such as books published in multiple languages. Instead of defining a unique type which contains the different language properties, the book description has one property with multiple values. For instance:

```
:x rdfs:label "German"@de .
:x rdfs:label "English"@en .
```

The fourth incompatibility problem occurred where there were URI predicates that referred to data which did not yet exist. Translating this information to SQL creates foreign keys to non-existent data in the database. This results in the creation of an illegal constraint. Since it is possible to query for such a URI in SPARQL, incorrect results are returned for the test case.

These four types of data incompatibility eliminated 68 of the DAWG cases from usability testing with relational databases. However, this was just the start of the incompatibilities; queries posed many more problems.

[4] http://www.w3.org/2001/sw/DataAccess/tests/#sort-3

Issuing the Queries

There are certain keywords that SBRD does not currently support: ASK, DESCRIBE, and ORDER BY. SBRD relies on Jena solely for navigating models, unlike D2RQ, so support for each of these keywords must be added where possible. This is an obvious shortcoming of SBRD, but nonetheless eliminated a good number of tests.

Several tests contain queries with variable predicates. While useful when querying a RDF graph, these translate poorly to relational databases. Querying using a variable predicate would cause a query to examine each table in the database as it searches for results. For instance, a user might ask for everything that has value “Foo.” This request would form a query that would select from each table and column containing string values and include a constraint specifying that the result must be “Foo.” For a relatively small database, this query would be lengthy. For a large database this query would be monstrous and require an extraordinarily long time to complete. Due to these potential problems, variable predicates are not allowed in queries sent to SBRD, and this restriction eliminated 21 DAWG test cases.

There are several tests that are beyond the scope of tools like D2RQ and SBRD. Specifically, there are tests that verify Unicode and internationalization support. This support is not the responsibility of the SPARQL to relational database adaptor since it is not reading in the query or storing the information. Along the same lines, there are negative tests that verify the SPARQL syntax parsing. Both D2RQ and SBRD rely on ARQ to parse the SPARQL query; therefore, if the query is malformed, an exception is thrown long before the query reaches the adaptor. Lastly, there are tests that load untrusted graphs and perform a query over these graphs. This circumvents the adapters completely, since the data is stored in a file not a database. Due to the nature of all these tests, they were eliminated.

Coverage

After each test was measured against the exceptions listed above, only 24 out of 170 tests were applicable to SBRD. With so few test cases available to test a relational database adaptor, it is impossible to determine if the adaptor is truly SPARQL compliant. Many of the test cases that are not usable can easily be changed so that they smoothly translate into a relational database environment without losing the concepts being tested. However, there are also tests that can never be translated into a relational database environment due to the limitations of relational databases. Below is a breakdown of incompatible tests. Some tests are counted more than once in this table since they fail for multiple reasons.

Incompatibility	**Number Failed**
Multiple data types for a single predicate	36
Untyped and typed data that should be in one table	1
Multiple Values for a subject predicate pair	23
URIs that refer to non-existent data	8
Unsupported keyword	50
Variable predicate	21
Test out of scope	115

Conclusion

For tools like D2RQ and SBRD, a new set of test cases that apply to relational databases must be created. These would be a highly customized subset of the current DAWG test cases. This subset would only verify the SPARQL functionality that is relevant to relational databases and the associated adaptors. All query types and keywords that make sense in a relational database environment would also be verified. Ideally, all data in the test cases should be appropriately typed so that representatively named tables can be created. All foreign keys should refer to another table. Lastly, all data in an individual column must be of the same type; this will change the design of tests such as type comparison and promotion.

Much of the groundwork for creating a set of test cases for SPARQL to relational database adaptors has already been laid. The tool used to convert the DAWG test cases' data to SQL is available at: http://projects.semwebcentral.org/projects/rdf2sql/. With the growing need to integrate relational databases into the Semantic Web, more tools like D2RQ and SBRD will be developed. Also, as the SPARQL specification changes, a W3C approved set of test cases for these tools should be developed.

Appendix A

DAWG Test Case: Optional-Filter[5]

Original RDF

```
@prefix x: <http://example.org/ns#> .
@prefix : <http://example.org/books#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

:book1 dc:title "TITLE 1" .
:book1 x:price  10 .

:book2 dc:title "TITLE 2" .
:book2 x:price  20 .

:book3 dc:title "TITLE 3" .
```

Generated SQL

```
CREATE TABLE Anon(
      ROW_URI VARCHAR(255),
      title VARCHAR(255),
      price INT,
      PRIMARY KEY(ROW_URI)
);
INSERT INTO Anon(
      ROW_URI, title, price
) VALUES(
      'http://example.org/books#book3',
      'TITLE 3',
      null
);
INSERT INTO Anon(
      ROW_URI, title, price
) VALUES(
      'http://example.org/books#book2',
      'TITLE 2',
      20
);
INSERT INTO Anon(
      ROW_URI, title, price
) VALUES(
      'http://example.org/books#book1',
      'TITLE 1',
      10
);
```

[5] http://www.w3.org/2001/sw/DataAccess/tests/#optional-filter