SQL, XQuery, and SPARQL

What's Wrong With This Picture?

Jim **Melton**

Abstract

Does the world really need Yet Another Query Language? A new language for querying RDF, named SPARQL, is emerging from the W3C. Some observers say that the W3C's own XQuery is sufficient for querying RDF, at least in its XML incarnations, while others suggest that SQL is a more mature, widely-implemented language for querying tuples. This presentation explores these issues and positions the three languages.

Table of Contents

1. Introduction

The World-Wide Consortium (W3C) has recently published two Last Call Working Drafts [SPARQL Language][SPARQL Results] and one Working Draft [SPARQL Protocol] defining a new query language named SPARQL. This new language is described as “a query language for getting information from ... RDF graphs” (that is, an RDF query language), which seems on the surface to be a new technology requirement.

But is it? RDF is described [RDF Concepts] as “a collection of triples, each consisting of a subject, a predicate and an object”. Triples are, of course, 3-tuples, and the well-known relational model of data is explicitly designed to represent tuples and collections of them. Similarly, SQL is a language expressly designed for the identification and retrieval of information from collections of tuples.

On the other hand, RDF can be (and frequently is) represented in XML [RDF Syntax]. Another language currently nearing completion within the W3C, XQuery [XQuery], has been carefully designed for the location and retrieval of information from XML documents.

A naïve user could be excused for wondering just why SPARQL is justified, given that two existing query languages—one of them developed in the same organization—appear to cover the requirements that led to the new language. In this paper, we examine the relationships between SQL, XQuery, and SPARQL to determine whether a new query language for RDF is justified.

2. Data Models

Query languages are typically designed to be applied to data corresponding to a particular data model. For example, SQL [SQL2003] is used to retrieve, create, modify, and delete data represented in (a variation of) the relational model of data. Similarly, XQuery is used to locate and retrieve (but not yet update) data that is represented in the XPath data model [DataModel].

It is sometimes possible to use one language to query data represented in a data model other than that for which the language was designed. This may be accomplished by *mapping* the data from its native data model into the query language's data model. One important example is a recent addition to the SQL standard, SQL/XML [SQLXML]. SQL/XML allows relational data to be *published* in an XML form (XPath data model instance) that can then be queried using XQuery. Additionally, it provides a facility called `XMLTABLE` that allows XML to be viewed as through it were ordinary SQL tabular data. Naturally, such mappings run into the famous “impedance mismatch” caused by factors such as the collections of data types differing amongst query languages and their corresponding data models.

RDF is presented as yet another data model, distinct from the XPath data model and from the SQL data model. It is tempting to reject that assertion because of the tuple nature of RDF entities. However, a close examination of [RDF Concepts] shows subtle differences between collections of RDF triples and multisets of rows in SQL tables of three columns. For example, SQL tables are defined to comprise one or more columns, each having a particular declared data type (such as `INTEGER`, `TIMESTAMP`, or some user-defined type). Every row in that table has exactly that number of columns and the value of each column in each such row must be of the column's declared type.

For columns of a user-defined type, values may have a *most-specific type* that is a subtype of that user-defined type. None of SQL's built-in types are defined to have subtypes (or supertypes), so this notion does not apply to columns declared to be of those types.

Notably missing from the definition of SQL tables is the idea that rows in a given table contain information about the data types of any of the (other) data in that table. SQL's *metadata* is recorded in a number of “system tables”, which (if actually provided by a given SQL implementation) could be combined in some way with the data in the table—although the criteria for such combinations to be made meaningful are unclear. By contrast, a given RDF collection can be augmented by RDF triples expressed using OWL [OWL Language] constructs that specify the *class* to which a

given RDF entity belongs. We are currently investigating whether the use of SQL's user-defined types would offer some way to map such class information from RDF into the SQL model.

Another difference that is sometimes important relates to the metadata associated with each model. In the SQL environment, the data literally cannot exist without the metadata—the schema. The two are inseparable in theory and in practice. However, in both the XPath data dodel and in the RDF data model, the data may exist independent of any schema describing the data. (Of course, the absence of a schema often limits the ways in which the data can be interpreted, but it is nonetheless possible to build XML documents and RDF collections without any schema that describes them.) On the World Wide Web, this distinction is especially important, because, unlike in the closed world of a database system, it is impossible for there to be a central point of control at which such metadata can be created...and enforced.

3. Strengths and Weaknesses

Each of these data models and corresponding query languages has advantages and disadvantages. SQL and the relational model are well designed to represent highly regular (“structured”) data such as that used by many business processes. Widely used examples include personnel and departments, students and classes, and manufacturing components. Such data usually includes a value for every column of every table. SQL (but not the pure relational model!) supports a special value—called the *null value*—to represent data that is missing, unknown, or inapplicable. The possibility of null values complicates the definition of and queries written in the SQL language, which makes SQL awkward for use in dealing with less structured information. The syntax of the SQL language focuses on identifying data for which most or all components are available and combining data based on the values of those components. In particular, combining data from two or more tables is specified by explicit SQL operators such as `JOIN` or `UNION`.

XPath, XQuery, and the XPath Data Model are all directed at support of less regular data. These languages and their data model function well when presented with data in which most or all of the values are present, but they also function well when applied to data in which many values are not represented at all. Such data, often called “semi-structured data”, is quite common in the XML tree-structured world in which elements and attributes may be optional and omitted entirely from instance data. It has been argued that the XPath Data Model represents a superset of, and could thus supersede, the relational model. We reject that conclusion because of the inherent overhead required by the XPath Data Model to self-identify each piece of data *versus* SQL's regular structures. That is, each datum in an SQL table takes its “name” from the name of the column in which it appears, while the XPath Data Model requires that the name of each datum be given explicitly—on every instance—as the name of the element or attribute of which it is the value. XQuery syntax is optimized for building new XML documents from one or more inherently semi-structured XML documents, easily accomodating the complete absence of some data. Combining data from two or more (source) XML documents is specified through the use of explicit operators such as a comma in a `for` expression or the keyword `union`.

To which of these camps does RDF belong? Every RDF triple comprises a subject, a predicate (or *property*), and an object, which—even though the subject or the object may not be explicit (that is, they may be represented by “blank nodes”) in some triples—implies that RDF is structured data. However, RDF defines a graph-based (in years past, the term “network” was often used) data model, which—like tree-based data models such as XML—supports the concept of optional data. In fact, [RDF Concepts] states that “it is not assumed that complete information about any resource is available”. We conclude that the RDF model is structured in the same sense that the relational model is: every provided assertion—SQL row or RDF triple—is complete (with the occasional missing datum represented by an SQL null value or an RDF blank node), but there may be assertions missing from the table or graph. As we show in Query Examples, SPARQL's syntax has been designed to combine information taken from one or more RDF graphs without the need for query authors to explicitly identify the mechanisms by which the graphs are combined. That is, operations such as joins are implicit rather than explicit in the language's syntax.

A major strength of the RDF model is that it has been extended with additional semantics in the form of OWL constructs. OWL permits a definition of *ontologies*, or semantics that are shared and agreed by multiple users. In ontologies specified in RDF and OWL, all known semantics of the domain being described are provided explicitly. In the XPath data model, such semantics are often provided only implicitly, such as in the sequence in which elements are permitted or

the parent-child relationships inherent in XML documents' structures. It is such ontologies that enable the semantic web and permit the application of machine reasoning on (that is, discovery of sometimes unpredicted relationships among) data.

4. Transformations

In many ways, RDF corresponds to an entity-relationship model. SQL tables—at least those with an explicit *key*—of *n* columns could be decomposed into *n*–1 entity-relationship assertions for each row, each such assertion having the form “key-value column-name column-value” (for example, “Srinivas salary 125000.00”). In fact, this observation provides a (trivial) method of transforming SQL tables into RDF graphs. Similarly, XPath Data Model instances could also be decomposed into entity-relationship assertions of the form “parent-element-node-id has-childor-attribute child-element-or-attribute-value-or-node-id” (*e.g.*, “node-for-Yungmin attribute-named-birthdate 1975-06-15”). Again, this provides a trivial method of transforming XML documents into RDF collections.

We mentioned earlier (Section 2, “Data Models”) that SQL/XML provides the ability to publish relational data into an XML form. That standard also provides a default mapping from SQL tables into XML. We call such mappings “lenses” because they permit applications to “see” XML when they “look at” relational data. The default mapping is necessarily highly generalized and does not account for applications' interpretation of the data being mapped. The preceding paragraph illustrates how such lenses could be built for mapping relational data and XML data into RDF. Those mappings are similarly naïve and do not necessarily capture the essence of the data's meaning. In all such cases, customized lenses can be constructed by application architects to perform mappings that capture more of the semantics of the data. Whether all semantics of the data can be transformed is an open question.

In the same way that data can be mapped, or transformed, between data models, queries written in the language associated with each such model can often be translated into the language associated with another model. For example, we know of at least one XQuery implementation that transforms XQuery expressions into SQL expressions to operate on XPath Data Model instances that have been transformed into SQL tabular data (through an operation commonly called “shredding”). Our exploration of this notion has demonstrated—but not yet *proved*—that it is similarly possible to translate (most?) SPARQL queries into SQL syntax. We suspect that SPARQL queries can also be translated into XQuery syntax, but have not yet investigated that possibility. It may also be possible to transform SQL expressions and/or XQuery expressions into SPARQL syntax, but we have not even started to look at that.

In Query Examples, we illustrate an SQL query and the corresponding XQuery and SPARQL queries.

5. Query Examples

To illustrate the different approaches taken by SQL, XQuery, and SPARQL, we provide “the same” data in three forms: relational tables (Table 1, “EMPLOYEES Table” and Table 2, “DEPARTMENTS Table”), two XML documents (Example 1, “Employees Document” and Example 2, “Departments Document”), and a collection of RDF triples (Table 3, “RDF for Employees and Departments”), then write “the same” query in each language. The query we have chosen, expressed in natural language, is: What is the salary of the manager of each department?

First, we provide abbreviated versions of two SQL tables describing employees and departments, as seen in Table 1, “EMPLOYEES Table” and Table 2, “DEPARTMENTS Table”, respectively.

ID	NAME	SALARY	DEPARTMENT
105	Jung-Shin Park	105000.00	Software
...	...	...	...
29	Ralph Swinden	91500.00	Marketing

Table 1. EMPLOYEES Table

ID	NAME	MANAGER
21	Software	105
...	...	...
83	Marketing	105

Table 2. DEPARTMENTS Table

The SQL query expression corresponding to our question is:

```
SELECT UNIQUE E.SALARY
FROM EMPLOYEES AS E JOIN DEPARTMENTS AS D
USING E.ID = D.MANAGER
```

Next, we illustrate abbreviated XML documents that describe employees and departments, as seen in Example 1, “Employees Document” and Example 2, “Departments Document”, respectively.

Example 1. Employees Document

```
<?xml version="1.0" encoding="UTF-8"?>
  <employees>
    <employee id="105">
      <name>Jung-Shin Park</name>
      <salary>105000.00</salary>
      <department>Software</department>
    </employee>
    <employee id="...">
      <name>...</name>
      <salary>...</salary>
      <department>...</department>
    </employee>
    <employee id="29">
      <name>Ralph Swinden</name>
      <salary>91500.00</salary>
      <department>Marketing</department>
    </employee>
  </employees>
```

Example 2. Departments Document

```
<?xml version="1.0" encoding="UTF-8"?>
  <departments>
    <department id="21" name="Software">
      <manager ref="105"</>
    </department>
    <department id="..." name="...">
      <manager ref=""</>
    </department>
    <department id="83" name="Marketing">
      <manager ref="105"</>
    </department>
  </departments>
```

The XQuery expression that states our question is:

```
for $e in
    fn:doc("http://employees.example.com/emps-doc.xml")/employees/employee,
  $d in
    fn:doc("http://employees.example.com/depts-doc.xml")departments/department
where $e/@id eq $d/manager/@ref
return $e/salary
```

Finally, we present an abbreviated RDF collection that describes employees and departments, as seen in Table 3, “RDF for Employees and Departments”.

In Table 3, “RDF for Employees and Departments”, we presume that the following namespace declarations are in effect:

```
emps:       http://emps.example.org/employees#
depts:      http://depts.example.org/departments#
rdb:        http://databases.example.org/
```

Subject	Predicate	Object
emps:id105	rdb:employees/column#name	Jung-Shin Park
emps:id105	rdb:employees/column#salary	105000.00
emps:id105	rdb:employees/column#department	depts:id21
emps:id29	rdb:employees/column#name	Ralph Swinden
emps:id29	rdb:employees/column#salary	91500.00
emps:id29	rdb:employees/column#department	depts:id83
emps:id...	rdb:employees/column#...	...
depts:id21	rdb:departments/column#name	Software
depts:id21	rdb:departments/column#manager	emps:id105
depts:id21	rdb:departments/column#name	Marketing
depts:id83	rdb:departments/column#manager	emps:id105
depts:id...	rdb:departments/column#...	...

Table 3. RDF for Employees and Departments

Our question expressed in SPARQL (assuming the same namespace declarations) is:

```
select ?salary
where
  {
    ?e rdb:employees/column#salary ?salary .
    ?d rdb:departments/column#manager ?e .
  }
```

6. So, What *Is* Wrong With This Picture?

Perhaps the most important thing that is “wrong with this picture” is the confusion that is likely to arise among potential users of the technologies discussed in this paper. We have already started to get questions from smart, well educated application developers asking for an explanation—a set of simple rules, if possible—about the circumstances under which they should choose to use a relational database and SQL, XML documents and XQuery, or RDF and SPARQL.

Under the assumption that people understand the tradeoffs between relational databases, XML documents, and RDF collections, it is somewhat easier to explain the justification for each of the three languages. But we continue to encounter people who ask why RDF isn't, or shouldn't be, stored in a relational database and then queried using SQL. Surely, they say, those “reasoning engines” that operate on RDF and OWL constructs could just as easily operate on them in the context of a relational database as in a new kind of collection. In that case, why wouldn't SQL be the language of choice for answering questions before and after those engines have done their jobs?

The best answer we have been able to provide is this: SPARQL syntax makes virtually all join operations implicit, while SQL syntax usually makes them explicit. A consequence of this design decision is that the SQL expressions to answer typical questions that will be asked against RDF collections tend to be much larger and somewhat more difficult to create (correctly!) because of the need to write explicit join operations and the explicit join conditions. Because typical questions made of RDF involve several, sometimes many, join operations, SPARQL provides a more compact notation that is perhaps easier to get right with less debugging time spent.

Similarly, we answer questions asking whether XQuery wouldn't be more appropriate, since RDF is typically serialized as XML, with a similar answer: the number of explicit join operations in `for` clauses and the number of join conditions

required in the `where` clauses probably makes the XQuery expressions more tedious to write and to debug than the corresponding SPARQL queries.

7. Conclusions

We have, somewhat reluctantly, concluded that the design goals of SQL and SPARQL are sufficiently different that there is adequate justification for the creation of a special-purpose language for querying RDF collections. We are comforted by the belief that it is possible to translate SPARQL expressions into SQL expressions, allowing users to store their RDF collections in relational databases if they wish to do so, and to write their queries in either SQL or in SPARQL, as they see fit. While predicting that it will be similarly possible to serialize RDF collections into XML documents and transform SPARQL expressions into XQuery expressions, we do not believe that most users would take that direction.

Acknowledgements

The author would like to thank Ashok Malhotra, Dana Florescu, and Jeff Mischkinsky for their help in researching topics related to the semantic web, including RDF and SPARQL.

Bibliography

[SPARQL Language] *SPARQL Query Language for RDF* *[http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050721/]*.

[SPARQL Results] *SPARQL Query Results XML Format* *[http://www.w3.org/TR/2005/WD-rdf-sparql-XMLres-20050801/]*.

[SPARQL Protocol] *SPARQL Protocol for RDF* *[http://www.w3.org/TR/2005/WD-rdf-sparql-protocol-20050527/]*.

[RDF Concepts] *Resource Description Framework (RDF): Concepts and Abstract Syntax* *[http://www.w3.org/TR/rdf-concepts/]*.

[RDF Syntax] *RDF/XML Syntax Specification (Revised)* *[http://www.w3.org/TR/rdf-syntax-grammar/]*.

[XQuery] *XQuery 1.0: An XML Query Language* *[http://www.w3.org/TR/xquery/]*.

[SQL2003] ISO/IEC 9075:2003, *Information technology — Database languages — SQL*, International Organization for Standardization2003.

[DataModel] *XPath 2.0 and XQuery 1.0 Data Model* *[http://www.w3.org/TR/2005/WD-xpath-datamodel/]*

[SQLXML] ISO/IEC 9075-14:2003*Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML)*, International Organization for Standardization2003.

[OWL Language] *OWL Web Ontology Language Reference* *[http://www.w3.org/TR/owl-ref/]*.

Biography

Jim **Melton**
Consulting Member of Technical Staff
Oracle Corporation [http://www.oracle.com]
Sandy
Utah
United States of America

Jim has been the editor for all parts of the SQL standard for SQL-92, SQL:1999, and SQL:2003, and has published five books covering various aspects of the SQL standard. Jim has represented his employer in the ANSI standards community and the United States in the international standards community for nearly two decades. A founding member of the SQLX Group, he is active in developing (as well as editing) a new part of the SQL standard, called SQL/XML, that specifies technology for effectively using SQL and XML together. In addition, Jim is a co-chair of the World Wide Web Consortium's XML Query Working Group. He is an editor of the two parts of the suite of documents being developed by the Query WG: Functions & Operators and XQueryX. Jim is working on a new book that deals with querying XML data in various ways.