Data Management between Relationalized Data and Semantic Web Data

Kyawt Kyawt San, Khin Nweni Htun
University of Computer Studies,Yangon
kyawtkyawts@gmail.com,knntun@gmail.com

Abstract

The RDF (Resource Description Framework) model has attracted the attention of the database community and many researchers have proposed different solutions to store and query RDF data efficiently. This paper proposes a framework for manipulating RDF data store and proposes a Data mapping algorithm. This paper also presents an approach to relationalize RDF data with the querying power of RDBMS and manipulate RDF data store using SPARQL/UPDATE. Existing approaches generally lack of focusing on updating RDF data store. We will show that despite its simple light-weight architecture, our system is able to outperform simultaneously in both retrieving relationalized RDF data and managing RDF data store in accordance with relationalized data.

1. Introduction

The Semantic Web is an effort by the W3C to enable integration and sharing of data across different applications and organizations. One area in which the Semantic Web community differs from the relational database community is in its choice of data model. The Semantic Web data model, called the "Resource Description Framework", or RDF, is a language for representing information about resources in the World Wide Web. RDF describes a particular resource using a set of RDF statements of the form (subject, predicate, object) triples, also known as (subject, property, value). The subject is the resource, the predicate is the characteristic being described, and the object is the value for that characteristic. These triples can then be stored in a relational database with a three-column schema. One of the clear advantages of the RDF data model is its schema-free structure in comparison to the entity-relationship model where the entities, their attributes and relationships to other entities are strictly defined. The storage of course can be just a file in any of the existing notations: RDF/XML, N-triples, Turtle, or Notation3 (N3). However, big amounts of data obviously require a database-based solution such as faster processing, ability to access needed part of data. At present, the processing of RDF/RDFS documents as databases is not efficient due to the lack of data synchronization between two data store: RDF data store and its relationalized data. Besides, query processing, optimization technologies and other important data management facilities such as concurrency control and recovery control which are commonly found in a Relational DBMS are not available in an RDF engine.

Storing RDF data in a relational database requires an appropriate table design. There are attempts to store RDF/RDFS documents in relational databases such as Jena2 [22], Sesame [5], Column-Store [18], and SW-Store [1]. These solutions generally center on a giant triples table, containing one row for each statement. This paper will combine the strengths and weaknesses of the above-mentioned research results to develop a data management system for RDF data store.

The key contributions of the paper are:

- An application-independent data mapping algorithm for storing RDF data in relational format.
- A framework for data management between RDF data store and Relationalized data store.

2. Related Work

R2D[15], a relational wrapper for RDF Data Stores, which aims to transform, at run-time, semi-structured RDF data into an equivalent normalized relational schema, thereby bridging the gap between RDF and RDBMS concepts and making the abundance of relational tools currently in the market available to the RDF Stores. Samizdat RDF Store [4], is an on-demand translation of RDF queries that allows mapping any relational data structure to RDF model, and perform queries over a combination of mapped relational data and arbitrary RDF triples with a performance comparable to that of relational systems.

Transformation engine [21] takes a different approach. In order to easily manipulate the database, RDF/RDFS documents are transformed into relational database format so that relational languages, data management and business intelligence facilities which are readily available can be exploited. A conceptual meta schema that describes RDF/RDFS documents and the corresponding meta table are presented together with illustrated examples. ONTOACCESS[3] that adds ontology-based write access to relational data. ONTOACCESS consists of the update-aware RDB to RDF mapping language R3M and algorithms for translating SPARQL/Update operations to SQL. BGPtoSQL[6], a basic graph pattern translation algorithm, that translates a basic graph pattern to its SQL-equivalent based on BGPtoSQL. In [7], an effective method to translate a complete SPARQL query into a single SQL is proposed, so that the generated SQL can be directly embedded as a sub-query into other SQL queries.

Sesame[5], an architecture for efficient storage and expressive querying of large quantities of metadata in RDF and RDF Schema. Sesame's design and implementation are independent from any specific storage device. Thus, Sesame can be deployed on top of a variety of storage devices, such as relational databases, triple

stores, or object-oriented databases, without having to change the query engine or other functional modules.

With regards to implementation, query languages such as SQL and SPARQL try to push as much of the functionality as possible to underlying tables storing RDF data. Our approach for implementing the relationalized RDF table function is somewhat similar. However, it is tightly integrated with the SQL engine and with the SPARQL/UPDATE function in SPARQL engine to be able to propagating updates to relational data store to maintain data synchronization between two data store in addition to the typical relationalizing approaches.

The remainder of this paper is organized as follows. The RDF data Model is presented in Section 3. In Section 4, propose a data mapping algorithm and introduce the framework for data management between RDF and RDB and. Section 5 concludes this paper with an outlook on future work.

3. RDF Data Model

An RDF model is also referred to as RDF graph, where each triple forms a <property> edge that connects the <subject> node to the <object> node. It is based on the following rules:

1. A Resource is anything that can have a URI (e.g. all Web pages, all Web images, all files accessible through ftp, etc.) http://www.ucsy.edu.mm/conferences/
2. A Property has a name and describes some relationship (e.g. Creator, Title, Subject, etc.)

The RDF data represented as a collection of <subject, property, object> triples, can easily be stored in a relational database. For example, the RDF classes and the triple instances are shown in Figure 1(a) and (b) respectively.

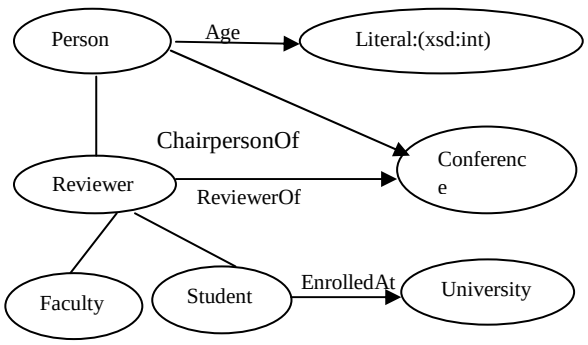

Figure.1 (a). RDF data for Reviewer Model

Subject	Property	Object
ICCA2011	Rdf:type	Conference
John	Age	24
John	rdf:type	Student
Mary	rdf:type	Faculty
Mary	Chairpersonof	ICCA2011

Figure.1(b). RDF data for Reviewer Model with Triple Instances

RDF tables are physically storing in a wider, flattened representation more similar to traditional relational schema. This flattened property table representation will require many fewer joins to access, since self-joins on the subject column can be eliminated. One can use standard query rewriting techniques to translate queries over the RDF triple store to queries over the flattened representation. In this paper, we adopt BGPtoSQL[6].

3.1 Simple Protocol And RDF Query Language (SPARQL)

SPARQL is the current W3C recommendation for querying RDF data. It is based on matching graph patterns against RDF graphs. A simple query can use SPARQL by obeying the some of the rules shown below:
PREFIX: Namespace definition
SELECT: constrains the output format (all obtained values for the variable ? name will be returned as a table)
WHERE: the query, as a graph pattern.
Variables: Start with ? or $

Example 3.1.

```
01SELECT ?name ?birthcountry ?number ?country
02 WHERE {
03 ?someone rdf:type :Person .
04 ?someone :name ?name .
05 ?someone :birthcountry ?birthcountry .
06 OPTIONAL {?someone :ssn ?number}
07 OPTIONAL {
08 ?someone :passportno ?number .
09 OPTIONAL { ?number :visacountry ?country }
10 }
11 }
```

In this example, WHERE clause contains both non-optional and optional parts. The non-optional part is the basic graph pattern defined with three triple patterns in lines 03-05. The basic graph pattern searches for the instances of class Person which have a name and a country of birth. The non-optional part must match for the query to succeed. Therefore, variables ?someone, ?name, and ?birthcountry must be bound.

The optional part includes three OPTIONAL clauses and does not have to match for the query to succeed.

3.2 Relationalizing RDF Data

Although there have been non-relational DBMS proposals for storing RDF data, the majority of RDF data storage solutions use relational DBMSs, such as Jena, Sesame, 3store. These solutions generally center on a giant triples table, containing one row for each statement.

RDF documents and RDF schemata can be considered at three different levels of abstraction:

i. at the syntactic level they are XML documents;
ii. at the structure level they consist of a set of RDF triples;

iii. at the semantic level they constitute one or more graphs with partially predefined semantics.

In this paper, we work at the structure level. Querying at this level means that any RDF model can be interpreted only as a set of triples, including those elements which have been given special semantics in RDF Schema. There are a number of architectural patterns that may be applied when developing a service for RDF-to-Relational data mapping. In this paper, we use a single table Triples (subject,predicate,object) to store RDF triples, such that each triple is naturally mapped to one row of the table. This triple store scheme, although not as efficient as some other storage schemas, is the best for our presentation purposes due to its simplicity and application independence.

For example, the RDF triples table for a small library dataset is shown in Table 1.

Table.1. Some sample RDF Triples

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"Semantic Services"
ID1	author	"H.Peter"
ID1	copyright	"2005"
ID2	type	CDType
ID2	title	"IELTS"
ID2	artist	"Judith Ash"
ID2	copyright	2003
ID2	Language	"English"
ID3	type	BookType
ID3	Title	Java
ID3	Language	English
ID4	Type	DVDType
ID4	Title	Matlab
ID5	Type	CDType
ID5	Title	"Office2003"
ID5	Copyright	2002

4. Architecture of Our Proposed Data Management System

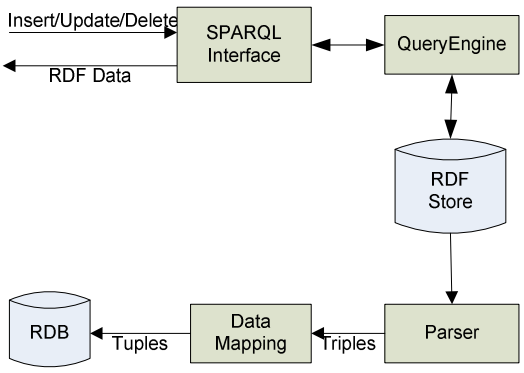

Figure.2. The architecture for relationalizing and manipulating RDF store

The architecture of a common RDF-to-Relational data mapping is shown in Figure 2.

4.1 Mapping Strategies for RDF

There are a number of architectural patterns that may be applied when developing a service for RDF-to-Relational data mapping. One such architectural pattern, used in this paper is Parser, which takes an RDF document as input and generates a set of RDF triples, each one consisting of a subject, predicate and object. The implementation of a parser may vary depending on the specific serialization format of an RDF document, such as the XML or N-Triples formats. Mapper is responsible for converting RDF triples into relational tuples that can be inserted into database tables. This component of the architecture is initialized with sufficient information about the database schema so that it can determine which table a triple should be inserted into and which columns the subject, predicate, and object belong to.

4.1.1 Proposed Data Mapping Algorithm

Our system resolve the conflict between the RDF data model and the target relational data model by proposing a mapping data mapping, is used to store RDF triples into relational tuples and insert them into the database.

A data mapping algorithm proposed in this paper is application independent.

01 Algorithm Data Mapping
02 Input: RDF Dataset D
03 Output: Dataset populated with relational tuples
04 Begin
05 Let RDFTriple' (sub, pred, obj) be a temp table
06 Parse D and load triples into RDFTriples' (sub, pred, obj)
07 For each resource in the RDFTriple'
08 Insert into RDFResource(resource_id, URI) ← select sub from RDFTriple (sub, pred, obj)
09 Insert into RDFPredicate(rdf_pred_id, URI) ← select pred from RDFTriple (sub, pred, obj)
10 Insert into RDFValuee(rdf_value_id, value) ← select obj from RDFTriple (sub, pred, obj)
11 End For
12 Insert into RDFTriple ← select sub_id, pred_id, obj_id from RDFResource, RDFPredicate, RDFValue
13 Delete all tuples from RDFTriple'
14 End Algorithm

Figure.3. Algorithm Data Mapping

4.2 Querying RDF Data Store

A number of query languages have been proposed and implemented that regard RDF documents as sets of such triples, and that allow querying such a triple set in various ways. The SPARQL Query Language is a W3C Candidate Recommendation for querying RDF, and as

such is fast becoming the standard query language for this purpose.

Efficiently querying RDF data is being an important factor in applying Semantic Web technologies to real-world applications. In this context, many efforts have been made to store and query RDF data in relational database using particular schemas.

4.3 Updating RDF Data Store

In order to make the Semantic Web real we need the infrastructure to store, query and update information adhering to the RDF paradigm. Such infrastructure can be developed from scratch or benefit from developments and experiences made in other science & technology realms such as within the database domain. For querying RDF data the WorldWideWeb Consortium released a Working Draft for the SPARQL query language. SPARQL/Update is a language to express updates to an RDF store. The approach is based on pushing as much work into the RDF store as possible in order to profit most from the SPARQL/UPDATE query techniques SPARQL/Update provides the following facilities:

- Insert new triples to an RDF graph.
- Delete triples from an RDF graph.
- Perform a group of update operations as a single action.
- Create a new RDF Graph to a Graph Store.
- Delete an RDF graph from a Graph Store.

The proposed version of SPARQL/Update consists of three update operations: (1)INSERT DATA (Listing 1) to insert new triples into an RDF graph; (2) DELETE DATA (Listing 2) to remove known triples from a graph; and (3) MODIFY (Listing 3) to delete and/or insert data based on triple templates that are matched against a triple pattern in a shared WHERE clause. The MODIFY operation basically corresponds to two SPARQL CONSTRUCT queries (with the same WHERE clause) where the resulting RDF triples get removed from and added to the data.

```
INSERT DATA { triples
}
```

Listing1: INSERT DATA

```
DELETE DATA { triples
}
```

Listing2: DELETE DATA

```
MODIFY
DELETE {
        Template
}
INSERT {
            Template
}
MODIFY {
Pattern
}
```

Listing3: MODIFY

4.4 Generating SQL Queries for Basic Graph Patterns

We adopt an Algorithm BGPtoSQL[6], is a primitive for translating a basic graph pattern into an equivalent SQL query, such that the SQL query retrieves RDF subgraphs matching the graph pattern from the triple store.

A basic graph pattern (BGP) is a set of triple patterns written as a sequence of triple patterns (separated by a period if necessary. A BGP should be understood as the conjunction of its triple patterns.

The SQL query result is a relation whose schema is the set of variables found in the graph pattern.

SPARQL is based on matching graph patterns against RDF graphs. In addition, SPARQL allows the specification of triple and graph patterns to be matched over RDF graphs.

The algorithm treats blank nodes as a special case of a variable with the scope of a basic graph pattern. Therefore, the algorithm substitutes every blank node label in the input graph pattern BGP with a unique variable, such that multiple occurrences of the same blank node are substituted by the same variable. The uniqueness property should hold for the scope of a SPARQL query to ensure that blank nodes in one basic graph pattern will not coincide with blank nodes in another pattern. All distinct variables in BGP are projected in the SELECT clause, such that a predicate/subject/object variable is represented by the corresponding column of the Triples table.

We would like to apply this algorithm for its efficiency and scalability.

Definition: Basic Graph Pattern Model

A basic graph pattern is modeled as a directed graph BGP = (N;E), where N is a set of nodes representing subjects and objects, and E is a set of edges representing predicates. Each edge is directed from a subject node to an object node. Each node is labeled (attribute label) with a variable name, a URI, a blank node, or a literal, and each edge is labeled with a variable name or a URI.

The SPARQL query in Example 1.1 has four basic graph patterns:

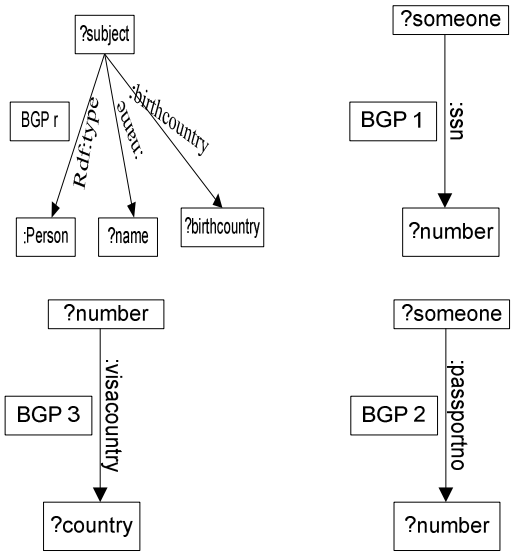

Figure.4. Basic graph patterns for Example 3.1

5. Conclusions

The paper proposed an application-independent data mapping algorithm to store RDF data in relationalized format. Specifically, a framework for data management between RDF data store and relationalized data store is introduced with the ability to maintain data consistency of these two data store. Wide adoption of the Semantic Web requires interoperability between relational databases and RDF applications. In this work, we designed an Relationalized RDF data management system for storing and querying RDF data. The described approach allows taking advantage of RDBMS transactions and the costs of migration from relational data model to RDF.

References

[1]A.J.Daniel, AdamMarcus, M.R.Samuel, H. Kate, "SW-Store: a vertically partitioned DBMS for SemanticWeb data management" The VLDB Journal (2009).

[2] "An Effective SPARQL Support over Relational Databases", Springer-Verlag Berlin Heidelberg 2008.

[3]Artem Chebotko, Xubo Fei, Cui Lin, Shiyong Lu, and Farshad Fotouhi , "Storing and Querying Scientific Workflow Provenance Metadata Using an RDBMS", Third IEEE International Conference on e-Science and Grid Computing.

[4] B. Dmitry, "On-demand RDF to Relational Query Translation in Samizdat RDF Store".

[5]B. Jeen, K.Arjohn, H.V.Frank, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema".

[6] C. Artem, Shiyong Lu, Hasan M. Jamil and Farshad Fotouhi, "Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns", Technical Report TR-DB-052006-CLJF, May 2006. Revised November 2006.

[7]Eric Prud'hommeaux, Alexandre Bertails,"A Mapping of SPARQL Onto Conventional SQL", World Wide Web Consortium (W3C), http://www.w3.org/

[8]Eugene Inseok Chong, Souripriya Das , George Eadon , Jagannathan Srinivasan, "An efficient SQL- based RDF Querying Scheme", Proceedings of the 31st VLDB Conference, 2005.

[9]F. BANCILHON and N. SPYRATOS, "Update Semantics of Relational Views ",

Franc¸ois Goasdou´e, Konstantinos, L.J. Karanasos, M. Ioana, "Materialized View-Based Processing of RDF Queries".

[10]H. Alice, B. Jeen, and S. Heiner, "RDF Storage and Retrieval Systems".

[11]H. Denis, "Semantic Web and RDF:Introduction". INRIA, France

[12]Jing Lu, FengCao, LiMa, YongYu, YuePan

K. Artem, "Storing and Querying RDF Data", Engineering (ITKS544), Spring 2009, University of Jyväskylä.

[13]M. Andrew, "Understanding SPARQL", (matthews.andrew@gmail.com)

[14]Matthias Hert, Gerald Reif, Harald C. Gall, "Updating Relational Data Via SPARQL/UPDATE".

[15]R. Sunitha, G. Anubha, K. Latifur, S. Steven, Bhavani, "A Framework for the Relational Transformation of RDF Data".

[16]RDF Primer, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[17]RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004

[18]S. Lefteris, G.Romulo, K.Martin, Column-Store Support for RDF Data Management: not all swans are white".

[19]S. Sherif and Ghazi Al-Naymat, "Relational Processing of RDF Queries: A Survey ".

[20] SPARQL Update**, "**A language for updating RDF graphs**",** W3C Member Submission 15 July 2008**,**http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/**.**

[21]T. Wajee and C. Suphamit, "A Transformation from RDF Documents and Schemas to Relational Databases".

[22]W. Kevin, S.Craig, K.Harumi, R Dave, "Efficient RDF Storage and Retrieval in Jena2".