A Framework for Querying Semantic Web Data Using SPARQL-SQL Query Translation

Thinn Thinn Win
University of Computer Studies, Mandalay
thinnthinn.win@gmail.com

Abstract

With the fast growth of Semantic Web, more and more Resource Description Framework (RDF) data are created and widely used in Web applications and information systems. Recently, efficient RDF data management on top of relational databases gains particular attentions from both Semantic Web community and database community. Most existing RDF stores, which serve as metadata repositories on the Semantic Web, use relational database management system (RDBMS) as a backend to manage RDF data. This paper presents a framework for querying the semantic web data over relational database. We also propose SPARQL-SQL (Simple Protocol and RDF Query Language - Structured Query Language) query translation algorithm that implements each SPARQL algebra operator via SQL query. And then, this translation algorithm generates a flat SQL statement for efficient query processing by relational database query engine, and their results can be returned as SPARQL query solutions.

1. Introduction

The Semantic Web has recently gained tremendous momentum due to its great potential for providing a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. Semantic annotations for various heterogeneous resources on the Web are represented in Resource Description Framework (RDF), and searched using the query language for RDF, called SPARQL. Essentially, RDF data is a collection of statements, called triples, of the form (s, p, o) where s is called subject, p is called predicate, and o is called object, and each triple states the relation between a subject and an object. Such a collection of triples can be viewed as a directed graph, in which nodes represent subjects and objects, and edges represent predicates connecting from subject nodes to object nodes. To query RDF data, SPARQL allows the specifications of triple and graph patterns that can be matched over RDF graphs.

SPARQL [5] is based on a graph matching facility, and supported by most RDF stores. Explosive growth of RDF data on the web has increased the need for novel RDF stores that can efficiently store and query large RDF datasets. Most existing RDF stores, including Jena [8], Sesame, RDFSuite, and Openlink Virtuoso [11] use a relational database management system as a backend to manage RDF data. This motivates us to study the problem of a complete translation from SPARQL queries into equivalent SQL queries, which can then be optimized and evaluated by a mature and vigorous relational query engine. It is crucial for many real-world Semantic Web applications to be able to access the content of non-RDF relational databases used by most legacy systems.

In this paper, we propose SQL model-based algorithm to (i) implement each SPARQL algebra operator via SQL query augmentation, and (ii) generate un-nested SQL statements for efficient processing by relational database query engines.

The rest of this paper is organized as follows. In Section 2, we review the previous work on querying Semantic Web data using an RDBMS,

in general, and on SPARQL-to-SQL translation, in particular. Section 3 presents preliminaries for our work. A Framework for Querying Semantic Web Data using SPARQL-to-SQL Translation is explained in Section 4. Section 5 reports our experiment demonstrating the effectiveness of our method. Section 6 concludes the paper.

2. Related Work

Harris et al. [14] presented an approach for translating SPARQL queries into SQL. They did not give a solution to the nested OPTIONAL pattern problem. A more sophisticated algorithm is required to express the nested optional graph patterns.

Cyganiak [12] presented a relational algebra for SPARQL assuming a schema-oblivious RDF store and outlines rules establishing equivalence between this algebra and SQL. This paper described a transformation from SPARQL into the relational algebra, an abstract intermediate language for the expression and analysis of queries.

Chebotko, et al. [2] proposed a semantics preserving SPARQL-to-SQL query translation algorithm for SPARQL queries that contain arbitrary complex optional graph patterns. They also [1] proposed provably semantics preserving SPARQL-to-SQL translation for SPARQL triple patterns, basic graph patterns, optional graph patterns, alternative graph patterns, and value constraints, and translation algorithm is generic and can be directly applied to existing RDBMS-based RDF stores.

Ma Li et al. [9] proposed a method to translate a SPARQL query into a single nested SQL, which can be directly used as a sub-query by other SQL queries. Each pattern node is translated into a SQL sub-query. Further, a facet-based scheme is designed to handle filter expression. However, it still has two limitations. First, the resulting nested SQL suffers from inefficiency. Second, their approach lacks the support for GRAPH node, named datasets, solution modifiers, and other query forms.

3. Preliminaries

We formalize the core fragment of SPARQL over RDF without RDFS vocabulary and overview of relational database and SQL.

3.1. RDF Data Model and SPARQL

The Semantic Web is an efficient way of representing data on the World Wide Web, or as a globally linked database and is generally built on syntaxes which use URIs(Uniform Resource Identifier) to represent data in triples based structures. RDF is a directed, labeled graph data format for representing information in the Web. This specification defines the syntax and semantics of the SPARQL query language for RDF.

Each arc in an RDF Model is called a statement. Each statement has three parts:
(i) subject - resource from which the arc leaves
(ii) predicate - property that labels the arc
(iii) object - resource or literal is pointed by arc

A statement is sometimes called a triple, because of its three parts. RDF Model is represented as a set of statements. Figure 1 shows the RDF Graph model (G).

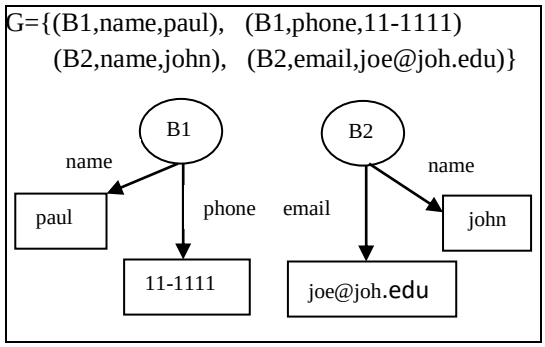

Figure 1.Sample RDF Graph

SPARQL [5] can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for

querying required and optional graph patterns along with their conjunctions and disjunctions.

3.2. Relational Database and SQL

The relational model provides a solid foundation to data consistency and its data structures are simple. They are two-dimensional tables whose elements are data items. The relational model of data is easier to formulate, and is now sizable and RDBMS has matured.

SQL is a declarative language that has been designed for querying and managing data stored in a relational database which can be regarded as a set of flat tables containing tuples. It is great for finding data from tabular representations and is designed to represent highly regular or structured data by many business processes.

4. A Framework for Querying Semantic Web Data using SPARQL-SQL Translation

The overview architecture of this system is shown in Figure 2 which visualizes the process of generating SQL code for a user's input. This system consists of tokenization, constructing SPARQL query operator tree, mapping relational algebra based on SPARQL, traversing query operator tree, SPARQL-SQL query translation and query processing. The parser converts the stream of input SPARQL query into a sequence of tokens and determines a relationship between the input tokens with a parse tree by applying syntactic analysis. This query is represented as query operator tree with a single root query operator that represents the query semantics. And then, we generate a flat SQL by applying proposed SPARQL-SQL translation algorithm based on the properties of SQL query model.

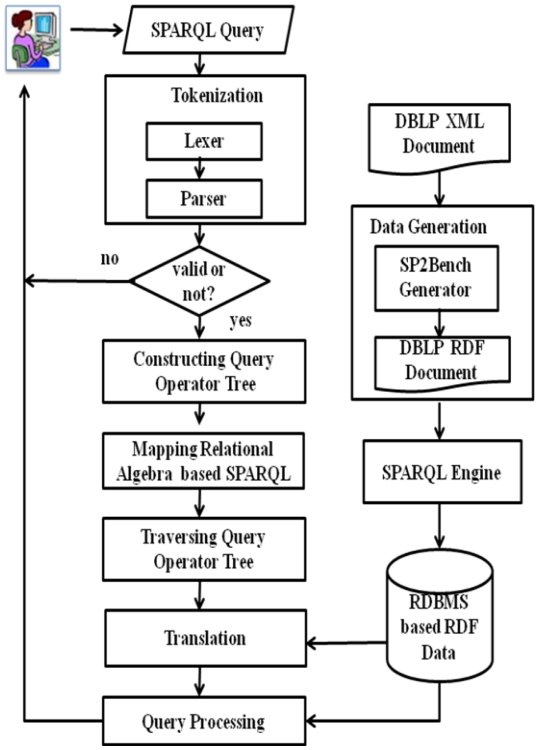

Figure 2.Overview Architecture of Querying Semantic Web Data over Relational Database

4.1. Tokenization

The first step of the translation is analyzing and parsing the user's input. The behavior of the lexer performs the lexical analysis which converts the stream of input characters into a sequence of tokens. The stream of tokens is passed to the parser in order to perform the syntactic analysis which determines a relationship between the input tokens. Tokens that are the atomic elements of SPARQL query syntax are shown in Figure 3.

Token	Description
"abc"	string
"abc"@en	string with language tag
123	specifically an xsd;integer
http://example.org/	IRI (or URI)
_:abc	blank node

?x	variable
ex:123	prefixed name
SELECT	symbol
+	symbol
@xyz	symbol

Figure 3.Token List Definitions of SPARQL

Figure 4 presents a simple SPARQL query example, which returns article and value. The WHERE clause in the example contains non-optional part. The non-optional part is the basic graph pattern defined with three triple patterns in line 3 and 4.

```
1.SELECT ?article ?value
2.WHERE {
3.      ?article rdf:type bench:article     tp1
4.      ?article swrc:pages ?value          tp2
  }
```

Figure 4. SPARQL Query Sample

The stream of tokens generated by the lexer is passed to the parser in order to perform the syntactic analysis which determines a relationship between the input tokens. The relationship can be visualized by means of a parse tree. Figure 5 shows a parse tree for a simple SPARQL SELECT query.

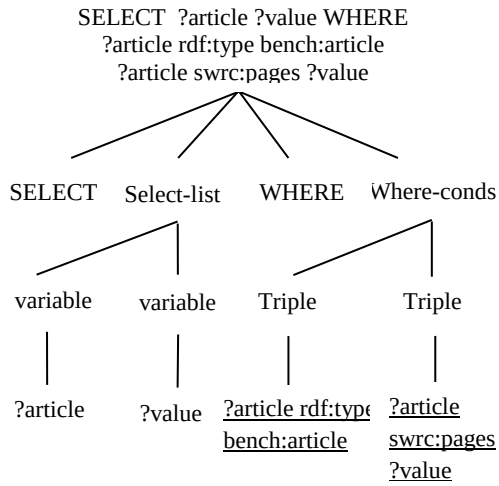

Figure 5.Parse Tree for SPARQL

4.1.1. EBNF Grammar Rules

The EBNF notation used in the grammar is defined in RDF. Keywords are matched in a case-insensitive manner. A parser decides which sentences are valid with respect to a given grammar. Some EBNF grammar rules are described as follows.

- SelectQuery::= 'SELECT' ('DISTINCT' | 'REDUCED')? (Var+|'*'DatasetClause* WhereClause SolutionModifier
- LimitClause::= 'LIMIT' INTEGER
- GroupGraphPattern ::= '{' TriplesBlock? ((GraphPatternNotTriples | Filter)'.'? TriplesBlock?)* '}'
- OptionalGraphPattern ::= 'OPTIONAL' Group GraphPattern
- Filter::= 'FILTER' Constraint
- Expression::= ConditionalOrExpression
- ConditionalOrExpression ::= ConditionalAnd Expression ('||' ConditionalAnd Expression)*
- ConditionalAndExpression ::= ValueLogical ('&&' ValueLogical)*
- BuiltInCall ::='STR'('Expression')'| LANG' ('Expression')' | 'LANGMATCHES' ('Expression', 'Expression')' | 'DATATYPE' ' ('Expression')'| 'BOUND' '(' Var ')' | 'sameTerm' ('Expression', 'Expression')' | 'isIRI ('Expression')' | 'isURI' 'Expression')' | 'isBLANK' ('Expression')'| 'isLITERAL' ('Expression')' | RegexExpression
- RegexExpression ::='REGEX' '('Expression', 'Expression' (',' Expression')?')'

4.2. SPARQL Pattern Tree

Generally, basic graph patterns in a SPARQL query can be expressed as a SPARQL pattern tree. The tree shows the backbone of the SPARQL query and is used in the translation. Similar pattern trees are used in [3]. Figure 6 illustrates the SPARQL pattern tree of the sample query. There are four types of nodes in the pattern tree: AND node, OR node, TRIPLE

node and FILTER node that represent graph patterns in the RDF graph.

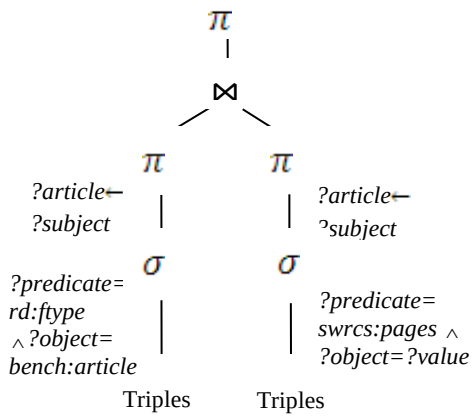

Figure 6. Relational Operator Tree of SPARQL Query

4.2.1. Syntax and Semantics of SPARQL

There are pairwise disjoint infinite sets I, B, and L (IRIs, Blank nodes, and RDF literals, respectively). A triple (v1, v2, v3) ∪ (I ∪ B) × I × (I ∪ B ∪ L) is called an *RDF triple*. In this tuple, v1 is the *subject*, v2 the *predicate* and v3 the *object*. We denote by T the union I ∪ B ∪ L.

4.2.2. Syntax of SPARQL Graph Pattern Expressions

In order to avoid ambiguities in the parsing, we present the syntax of SPARQL graph patterns in a more traditional algebraic way, using the binary operators UNION, AND, OPT, and FILTER. We fully parenthesize expressions and make explicit the left associativity of OPTIONAL and the precedence of AND over OPTIONAL implicit.

A SPARQL graph pattern expression is defined recursively as follows:
(i) A tuple from (T ∪ V) × (I ∪ V) × (T ∪ V) is a graph pattern (a *triple pattern*).
(ii) If P1 and P2 are graph patterns, then expressions (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are graph patterns.
(iii) If P is a graph pattern and R is a SPARQL *built-in* condition, then the expression (P FILTER R) is a graph pattern.

4.3. Relational Algebra based Semantics of SPARQL

We define the relational representation of SPARQL query solution. Let a tuple r: IVL → IBL ∪ {NULL}be a total function, that assigns RDF terms of an RDF graph to URIs, literals, and variables of a SPARQL query, where NULL denotes an undefined or unbound value. Then, the relational representation of a SPARQL query solution is a set R of tuples r or simply a relation R. The schema of R, denoted as $\xi(R)$, is the subset of IVL over which each tuple r ∈ R is defined; abusing the notation, we denote a tuple schema as $\xi(r)$ and $\xi(r) \equiv \xi(R)$ for all r ∈ R. We define R as an infinite set of all possible relations, each of which represents a SPARQL query solution.

R, R1, R2, and R3 denote relations, $\xi(R)$ denotes the schema of a relation R, ⋈ denotes an inner join, ⊎ denotes an outerunion, / denotes a set difference, and ρ,σ, and π denote renaming, selection, and projection operators of the relational algebra, respectively.

4.4. Traversing Query Operator Tree

A SPARQL query can be modeled as a query algebra operator tree, with a single root operator that determines the query output type and a set of nested basic query operators that represent the query semantics. Except for the root operator, every operator has a parent operator, parent (q). Operators that have child operators are called non-terminal operators, which have a common interface for accessing all child operators, children (q).

4.4.1 Required & Available Variables

The required variables for a query operator q, required (q), are variables that are needed for proper evaluation of the operator. There are four cases where variables are required: (i) the SELECT clause to specify a variable that will be in the query result, (ii) an ORDER BY clause, (iii) a Filter expression, and (iv) join sub-queries in a Join or Optional operator.

Select: required (q) = { v | v ∈ projectVars(q) orderBy(q)}

Filter:required(q) = {v | v ∈ vars(expr(q)) v ∈ required parent(q))}

Join/Optional: required (q) = {v | v ∈ available (lef(q) ∩ available(right(q))}

Other operator: required (q) = {v | v ∈ required (parent (q))}

The set of required variables are passed down the operator tree from the initial operator that set the requirement to their child operators. The available variables for a query operator q, available (q), are variables for which the source of a binding has been introduced through the evaluation of the current operator or through the evaluation of one or more child operators. Depending on the type of operator q, there are three cases:

(i) Triple Pattern or Graph: available (q) = {v | v ∈ vars (q)}

(ii) Other terminal operator: available (q) = ∅

(iii) Non- terminal operator: available (q) = {v | ∃ c (c ∈ children (q) ^ v ∈ available (q)))}

4.5. Properties of SQL Model m

This system presents a SQL model *"m"* for generating SQL queries. More specifically, we focus on SELECT…FROM…WHERE… ORDER BY queries with support for DISTINCT, LIMIT, and OFFSET modifiers. The model *m* allows join types to be specified along with join ON conditions. Nested SQL queries can be used in FROM and WHERE clauses and table and column aliases are supported as well. The properties of the model are given in table 2.

Property	**Description**
Project Cols(m)	List of column projection strings for SELECT clause
Tables(m)	List of tables used by the query with join types, join conditions, and table aliases for the FROM clause. For a table t in tables(m): - tableDef(t): name of table or nested SQL query - alias(t): alias of table - joinType(t): type of join with previous table -joinCond(t): boolean join conditions
Where(m)	Set of boolean conditions as strings to appear in conjunctive form in the WHERE clause.
OrderBy (m)	List of column names to use for ordering the results, with sort direction for the ORDER BY clause. For an order by item i in orderBy(m): - name(i): column name
Distinct (m)	Boolean value, true if only distinct results are returned.
Limit(m)	Integer value specifying the maximum number of tuples
Offset(m)	Integer value specifying the number of tuples

Table 2. Properties of SQL Model m

4.5.1. Proposed SPARQL-SQL Translation Algorithm

Now, we present the procedure for generating efficient SQLs for the SPARQL algebra operators. The fundamental idea of our approach is to implement the algebra operators by augmenting the SQL query that was constructed

by the parent algebra operator, instead of inserting a nested SQL subquery. We generate a flat SQL by applying proposed SPARQL-SQL translation algorithm by means of SQL model properties.

```
Input: SQL model m
Output: A SQL query string based on model m, insertion into a parent SQL query
1. Begin
2.  Assign "SELECT"  to sql
3.  If distinct(m) is true Then
4.   Add  "DISTINCT " to sql
5.  For each c in projectCols(m)
6.   Add "FROM" to sql
7.  For each t in tables(m)
8.    If first is True Then
9.     Assign False to first
10.    Add "tableDef(t) AS alias(t)" to sql
11.   Else
12.     "joinType(t) tableDef(t) AS alias(t) ON (joinCond(t)) " to sql
13.  If where(m) is not null Then
14.    Add "WHERE "to  sql
15.  For each c in where(m)
16.    Add "AND " to sql
17.   If orderBy(m) is not null Then
18.     sql+= "ORDER BY "
19.   For each o in orderBy(m)
20.     Add name(o) to sql
21.     If limit(m)is not NULL Then
22.       Add "LIMIT " + limit(m) to sql
23.     If offset(m) is not  NULL Then
24.       Add "OFFSET " + offset(m) to sql
25.Return sql
26.End
```

Figure 7. SPARQL-SQL Translation Algorithm

And then, we generate an equivalent SQL query with sample SPARQL query that is shown in Figure 4.The generated SQL query describes in Figure 8.

```
Select tp1.member As article,
       tp2.object As value
From Types tp1 Inner Join  LiteralProperties
tp2 On (tp1.member=tp2.subject or
         tp1.member is Null or
         tp2.subject is Null)
    Where tp1.class="bench:article"
    And tp2.precidate="swrc:pages"
```

Figure 8. SQL Query

5. Experiments

Our proposed method has been implemented existing SPARQL parser to convert a SPARQL syntax tree into a SPARQL algebra operator tree. From the operator tree, appropriate SQL was generated and executed against a MySQL database. In our experiments, we primarily compare the nested SPARQL-to-SQL approach proposed by Chebotko et al. [1], with their proposed SQL simplifications, and then with our SPARQL-to-SQL approach.

RDF scheme basically follows the RDF encoding approach which provides an XML-to-RDF mapping of the original DBLP data set is shown in Figure 9.

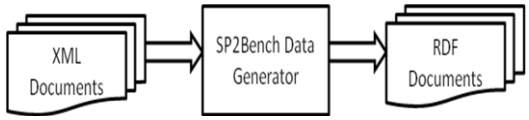

Figure 9. Mapping from DBLP XML File to RDF

DBLP dataset is generated using the data generator included in the SP^{2}Bench (SPARQL Performance Benchmark) [10]. In particular, the SP^{2}Bench is designed to test the most common SPARQL constructs, operator constellations, and the broad range of RDF data access patterns.

Moreover, the SP^{2}Bench is a data generator for creating DBLP-like RDF documents, which mirror characteristics and relations found in the original DBLP bibliography. Figure 10 illustrates the sample instances of DBLP RDF graph.

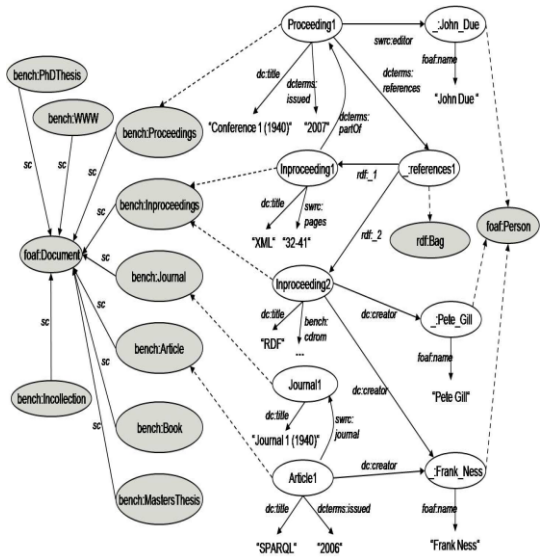

Figure 10. DBLP Sample Instances RDF

For the purpose of this experiment, a data set with 5 million RDF triples was generated for analysis. The SQL queries resulting from the translations will be executed against MySQL 5.0 database using Jena.

6. Conclusion

The SPARQL-SQL query translation is very important to bridge the two worlds of the Semantic Web and relational databases. This system describes the relational algebra based semantics of SPARQL. We propose the query translation algorithm that represent each SPARQL algebra operator via SQL query and generate efficient SQL queries. This translation frequently uses SQL features whose evaluation is not yet optimized by a relational database engine and generates to be simpler and more efficient SQL queries.

References

[1] A. Chebotko, S, Lu and F, Fotouhi, Semantics Preserving SPARQL-to-SQL Translations, Technical Report, 2009.

[2] A. Chebotko, S. Lu, H.M. Jamil, F. Fotouhi, Semantics preserving SPARQL-to-SQL query translation for optional graph patterns, Technical Report, Wayne State University, May 2006

[3] B. Elliott, E. Cheng, C. Thomas-Ogbuji, Z. Ozsoyoglu. A Complete Translation from SPARQL into Efficient SQL ,ACM, ,2009.

[4] D. Abadi,J., Marcus, A., Madden, S.R. and Hollenbach, K. Scalable Semantic Web Data Management Using Vertical Partitioning. In Proc. of VLDB, 2007, 411-422.

[5] E. Prud'hommeaux, and A. Seaborne, SPARQL Query Language for RDF. W3C, 2008. http://www.w3.org/TR/rdf-sparql-query/

[6] http://www4.wiwiss.fu-berlin.de/dblp/

[7] J. P´erez, M. Arenas and C. Gutierrez. Semantics and complexity of SPARQL. In Proc. of ISWC 2006. pp. 30-43.

[8] K. Wilknson, C. Sayers, H. Kuno and Reynolds, D. Efficient RDF storage and retrieval in Jena2. International Workshop on Semantic Web and Databases, 2003, 131-150.

[9] L. Ma, C. Wang, J. Lu and Y. Yu. Effective and Efficient Semantic Web Data Management over DB2, ACM SIGMOD, 2008, 1183-1194.

[10] M. Schmidt and T. Hornung. "SP2Bench: A SPARQL Performance Benchmark", 2008.

[11] O. Erling. Implementing SPARQL compliant RDF triple store using SQL-ORDBMS. OpenLink Software Virtuoso. 2001.

[12] R, Cyganiak. A relational algebra for SPARQL. Technical Report, HP Lab Bristol.

[13] R. Lu, F. Cao, L. Ma, Y. Yu and Y. Pan, An Effective SPARQL Support over Relational Databases, In Proc. Of VLDB 2007 workshop on Semantic Web, Ontologies, Databases, 2007.

[14] S. Harris and N. Shadbolt. SPARQL query processing with conventional relational database systems, SSWS, 2005.