Semantics Preserving SQL-to-SPARQL Query Translation for Left Outer Join

BAHAJ Mohamed, Soussi Nassima
Faculty of Science and Technologies, Settat
Morocco
mohamedbahaj@gmail.com
sossinass@gmail.com

ABSTRACT: *Despite the emergence of the Semantic Web and its performance, relational databases are still the most used for data manipulation, making the mapping of the relational world to the semantic world a very pertinent research topic to fill the gap between these two heterogeneous systems and ensure better semantic interoperability without physical transformation of data represented with different syntaxes. Regarding the transformation of SQL queries to SPARQL queries semantically equivalent, several algorithms have been developed to ensure this conversion; unfortunately, most of these algorithms have shortcomings and weaknesses in the consideration of the Left Outer Join commands in SQL queries. This problem has motivated us to write this paper so as to remedy this lack. Our main contribution is as follows: (a) establish an effective semantic transformation algorithm for Left Outer Join commands (in their simple and nested structures) to their equivalent in the SPARQL language, (b) integrate this result to RETRO (A Framework for Semantics Preserving SQL-to-SPARQL Translation) in order to have a richer algorithm that contribute to treatment of a maximum of SQL commands.*

Keywords: Semantic Web, SQL, SPARQL, Simple Left Outer Join, Nested Left Outer Join, RETRO

Received: 29 May 2017, Revised 3 July 2017, Accepted 10 July 2017

1. Introduction

The Semantic Web is a mesh of information linked up in such a way as to be easily processable by machines, on a global scale [1]. This is an efficient way of representing data on the World Wide Web. It is considered as a gateway to access the data between different applications and systems. These data are exchanged on the web via a standard model called RDF (Resource Description Framework) [3], it is a Framework allowing these data to be mixed, exposed and shared between different applications via a simple and powerful data model [4]. SPARQL [13, 2] is officially recommended by the W3C as the most representative description language for RDF data.

The major goal of the Semantic Web is summarized in the awarding of semantics to existing data on the web to have a linked data network exploitable by various applications. Currently, the most information is stored in relational databases, which makes the development of methods and tools for automatic conversion of relational databases in the semantic web a relevant need. Many approaches have been developed for converting SQL queries to SPARQL queries in order to make the contents of a relational database available in RDF stores, but unfortunately, most of these algorithms have weaknesses and gaps in taking into account the Left Outer Join command in SQL queries. This problem has motivated us to write this paper so as to remedy this gap and establish an algorithm which converts SQL queries of the types LOJ (Left Outer Join) to OPTIONAL queries semantically equivalent, integrating our result to the famous RETRO approach [5] in order to have at the end a richer Framework that can process the maximum cases. To our best knowledge, this paper is the first work developed in this direction treating the conversion problem of LEFT OUTER JOIN orders under their simple and nested structures.

The rest of the paper is organized as follows: Section 2 exposes some existing approaches in this topic. Section 3 describes our main contribution via an effective and detailed algorithm. The paper is concluded in section IV that also presents suggestions for future extensions of this approach.

2. Related Works

Several researchers [6, 7, 8, 9, 10, 12] have been developed taking into account the LOJ commands in the conversion direction of SPARQL queries to SQL queries. The paper [6] formalizes a relational algebra based semantics of SPARQL, which bridges an important gap between the Semantic Web and relational databases, and prove that this semantics is equivalent to the mapping-based semantics of SPARQL; Based on this semantics, the authors propose the first provably semantics preserving SPARQL-to-SQL translation for SPARQL triple patterns, basic graph patterns, optional graph patterns, union graph patterns, and value constraints. The authors in [7] propose a system called Ultrawrap with a novel architecture to representing relational data as RDF triples using unmaterialized views and which can answer SPARQL queries by translating them to SQL queries over the views. The work described in [8] present Left Bit Right (LBR), a technique for well-designed nested BGP and OPTIONAL pattern queries. Through LBR, we propose a novel method to represent such queries using a graph of supernodes, which is used to aggressively prune the RDF triples, with the help of compressed indexes. The paper [9] documents a semantic for expressing relational data as an RDF graph and an algebra for mapping SPARQL SELECT queries over that RDF to SQL queries over the original relational data. In [10], the authors propose a basic graph pattern translation algorithm which translates a basic graph pattern to its SQL equivalent; and based on this algorithm, a semantics preserving SPARQL-to-SQL query translation algorithm, SPARQLtoSQL, for SPARQL queries that contain arbitrary complex optional graph patterns. The authors in the paper [12] formalize a relational algebra based semantics of SPARQL; Based on this semantics, they propose the first provably semantics preserving SPARQL-to-SQL translation which takes into account even the optional graph patterns.

On the other hand, various approaches [5,11] have been made to guarantee the transition from relational databases to RDF data stores by adopting algorithms for conversions of SQL queries to SPARQL queries semantically equivalent. To our best knowledge, there is no published algorithm in this direction addressing the commands Left Outer Join, and this is what motivates us beyond all other considerations to write this paper in order to overcome this problem.

3. Query Mapping

Before starting the description of the transformation algorithm, we show with some examples how our strategy for the mapping of Left Outer Join commands to OPTIONAL clauses works. Then we describe all procedures of mapping queries algorithm: *MappingLeftOuterJoinQuery,CounterNbrLOJ* and the updated version of *Query-Mapping* of RETRO approach.

1. Examples

The table below describes a set of LOJ queries using SPARQL and SQL.

Generally speaking, the FROM clause containing the LEFT OUTER JOIN command can be represented as a binary tree, more precisely, unbalanced tree as shown in the figure below using the following abbreviations:

LOJ : LEFT OUTER JOIN command,

	Requête SQL	Requête SPARQL

Simple Left Outer Join Clause	SELECT t1.b as b, t2.c as c
	FROM (SELECT s as a, o as b FROM rdf_p1) t1
	LEFT OUTER JOIN
	(SELECT s as a, o as c FROM rdf:p2) t2
	ON (t1.a = t2.a)
		SELECT ?b ?c
		WHERE {
		?a rdf:p1 ?b
		OPTIONAL {
		?a rdf:p2 ?c
		}

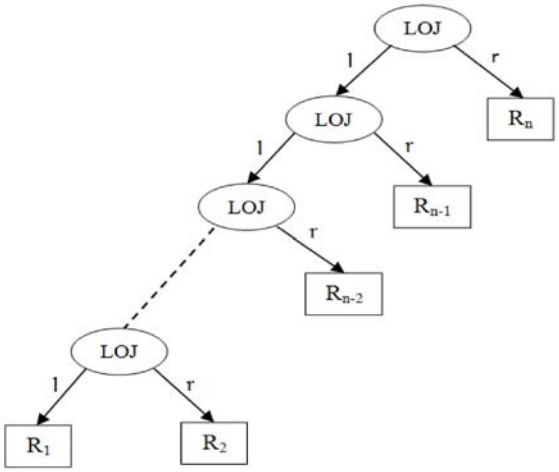
Nested Left Outer Join Clause	SELECT * FROM
	(SELECT t1.a as a, t1.b as b, t2.c as c
	FROM (SELECT s as a, o as b FROM rdf:p1) t1
	LEFT OUTER JOIN
	(SELECT s as c, o as a FROM rdf:p2) t2
	ON (t1.a = t2.a)
	) t3
	LEFT OUTER JOIN
	(SELECT s as c, o as d FROM rdf:p3) t4
	ON (t3.c = t4.c AND t3.a IS NOT NULL)
		SELECT ?a ?b ?c ?d
		WHERE {
		?a rdf:p1 ?b .
		OPTIONAL {
		?c rdf:p2 ?a .
		OPTIONAL {
		?c rdf:p3 ?d .
		}
		}
		}

Table 1. LOJ queries using SPARQL and SQL

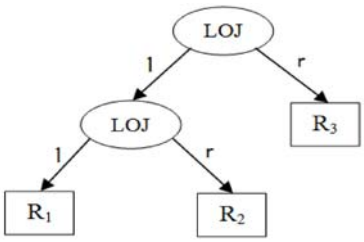
R_n : the relation (sql query) of rank n,

r : right,

l : left.



If we take for example the second SQL query (nested Left Outer Join clause) of the previous table, the binary tree of this query will be as follows:



With:

R_1 = SELECT s as a, o as b FROM rdf:p1

R_2 = SELECT s as c, o as a FROM rdf:p2

R_3 = SELECT s as c, o as d FROM rdf:p3

2. Transformation Algorithm

The conversion algorithm of the famous RETRO approach [5] is taken into account in this paper as a base on which we built our contribution in this direction (SQL-to-SPARQL) which treats the SQL join type Left Outer Join in its simple and nested form.

Based on the analysis of the previous examples, we developed an algorithm transforming LOJ commands to an Optional clauses semantically equivalent (*MappingLeftOuterJoinQuery*). This sub-procedure is called in the main procedure *Query-Mapping* from RETRO [5] in order to treat the LOJ SQL queries. Our sub procedure takes as input an SQL query (q_{in}) that is subsequently parsed to a binary tree T. Firstly, the algorithm traverses the tree T starting with the root (the main clause LOJ) so as to store the right relation of each node LOJ in a stack for further processed; secondly, it traverses the left side of the tree T until the detection of the left leaf that will represent subsequently the first basic graph pattern in the WHERE clause of the equivalent SPARQL query. As long as the stack is not empty, then the conversion process to SPARQL queries continues in the WHILE loop to extract the basics graph patterns which will be the equivalent Optional commands in the WHERE clause, as well as the variables of equivalent SELECT clause. The *cpt* variable is incremented in the WHILE loop at each iteration in order to count the number of OPTIONAL commands that will be used later in the conception of the final query so as to close the braces of each OPTIONAL command. The *NbrLOJ* variable represents the number of LOJ nodes in the tree T, it is counted via the sub procedure *CounterNbrLOJ()* that traverses the tree by counting all nodes except leaves.

Algorithm 1: MappingLeftOuterJoinQuery()

```
Input: A left outer join query, q_in
Output: An SPARQL query, q_out
Begin
Stack stack = EmptyStack
Select = ø {Map of Select variables initially blank}
Where = "" cpt=0
V_l = ø {Map of Select variables of left relations initially blank}
V_r = ø {Map of Select variables of right relations initially blank}
q_out = ""
Tree T = parse(q_in)
NbrLOJ = CounterNbrLOJ(T)
   for i ← 1 to NbrLOJ do
        if (i=1) then
            LOJ_i = root(T)
        else
            LOJ_i = LOJ_i-1.LeftChild()
       end if
       R_r = LOJ_i.RightChild()
       R_l = LOJ_i.LeftChild()
       stack.push(R_r)
            if (R_l.isNotLOJclause() = true) then
```

```
                q_sql = R_1
                q_sparql = Query_Mapping(q_sql)
                tree_1 = parse(q_sparql)
                BGP_1 = tree_1.getBGP()
                V_1 = tree_1.getSelectVar ()
               Select = V_1
                Where += BGP_1
          end if
     end for
     while (stack.isNotEmpty) do
                q_sql = stack.pop()
                q_sparql = Query_Mapping(q_sql)
                tree_r = parse(q_sparql)
                BGP_r = tree_r.getBGP()
                V_r = tree_r.getSelectVar ()
                V_sel = Select
                Select = V_sel *" (V_r  - (V_sel )" V_r))
                Where += 'OPTIONAL{' + BGP_r
                cpt = cpt+1
   end while
     q_out += "SELECT" + Select + "WHERE{" + Where
     for i  ← 1 to cpt.size do
          q_out += "} "  /* Cloture de chaque clause OPTIONAL */
     end for
          q_out += "}" /* Cloture de la clause WHERE */
End Algorithm
```

Algorithm 2: CounterNbrLOJ()

```
Input: A tree T
Output: A number of Left Outer join, NbrLOJ
Begin
     Stack ST = EmptyStack
     r = Root(T)
     ST.push(r)
     NbrLOJ = 1
     while (ST.isEmpty = false) do
          n = ST.pop()
          if (LeftChild(n) != NULL && LeftChild(n).isLeafNode()=false) then
               NbrLOJ = NbrLOJ + 1
               ST.push(LeftChild(n))
          end if
```

```
end while
return NbrLOJ
End Algorithm
```

Algorithm 3: Query-Mapping()

```
Input: An SQL query, q_in
Output: A SPARQL query, q_out
q_out =”” {A SPARQL query that is initially blank}
tree = parse(q_in) { A parse tree obtained by parsing q_in }
q_in^SELECT = tree.getSelectClause();   q_in^FROM = tree.getFromClause()
q_in^WHERE-JC = tree.getJoinConditions()
q_in^WHERE-BE = tree.getBooleanExpr()
q_out^SELECT = “SELECT”   q_out^WHERE = “WHERE {“
      TP ← ø { The map of triple patterns is initially empty}
      if tree.type = CrossProductQuery then
      T P = TransSQLFromClause(q_in^FROM)
      q_out^WHERE += TransSQLWhereClause(q_in^WHERE-JC, q_in^WHERE-BE ,TP, P)
      q_out^SELECT += TransSQLSelectClause(q_in^SELECT, TP)
      q_out = q_out^SELECT + q_out^WHERE +” }”
else if tree.type = SPJQuery then
      T P = TransSQLFromClause(q_in^FROM)
      q_out^WHERE += TransSQLWhereClause(q_in^WHERE-JC, q_in^WHERE-BE ,TP, P)
      TP = extractTP(q_out^WHERE)
      q_out^SELECT += TransSQLSelectClause(q_in^SELECT , TP)
      q_out = q_out^SELECT + q_out^WHERE +” }”
else if tree.type = UnionQuery then
      q_1 = tree.leftSubTree(), q_2 = tree.rightSubTree()
      q_1^out = Query-Mapping(q1), q_2^out = Query-Mapping(q2),
      q_out = Merge(q_1^out, q_2^out, “UNION”)
else if tree.type = IntersectQuery then
      q_1 = tree.leftSubTree(), q_2 = tree.rightSubTree()
      q_1^out = Query-Mapping(q_1), q_2^out = Query-Mapping(q_2),
      q_out = Merge(q_1^out, q_2^out, “INTERSECT”)
else if tree.type = ExceptQuery then
      q_1 = tree.leftSubTree(), q_2 = tree.rightSubTree()
      q_1^out = Query-Mapping(q_1), q_2^out = Query-Mapping(q_2)
      q_out = Merge(q_1^out, q_2^out, “EXCEPT”)
else if tree.type = LOJQuery then
      MappingLeftOuterJoinQuery()
end if
return q_out
```

4. Conclusion

In this paper, we have contributed in interoperability between RDBMS and RDF Stores as discussed above via the elaboration of an efficient algorithm for conversion of simple and nested Left Outer Join SQL queries to Optional SPARQL queries. This approach is very helpful because – to our best knowledge – there is no work to date that treats this issue.

One obvious extension of our research regarding SQL to SPARQL transformation for left outer join queries will be an improvement of our converter algorithm in order to support right and full outer joins.

References

[1] http://infomesh.net/2001/swintro/.

[2] SPARQL Query Language for RDF. W3C Candidate Recommendation, 15 January 2008.

[3] RDF 1.1 Concepts and Abstract Syntax.W3C Recommendation 25 February 2014.

[4] http://www.w3.org/RDF/

[5] Rachapalli, J., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B. (2009). RETRO: A Framework for Semantics Preserving SQL-to-SPARQL Translation, The University of Texas at Dallas, 800 West Campbell Road, Richardson, TX 75080-3021, USA, .

[6] Chebotko, Artem., Lu, Shiyong., Fotouhi, Farshad. Semantics Preserving SPARQL-to-SQL Translation, Department of Computer Science Wayne State University, 5143 Cass Avenue, Detroit, Michigan 48202, USA.

[7] Juan, Sequeda, F., Daniel., Miranker, P. (2013). Ultrawrap: SPARQL Execution on Relational Data, Department of Computer Science, University of Texas at Austin.

[8] Atre, Medha. Left Bit Right: For SPARQL Join Queries with OPTIONAL Patterns (Left-outer-joins), Pune, India.

[9] Prud'hommeaux, Eric., Bertails, Alexandre. A Mapping of SPARQL Onto Conventional SQL, World Wide Web Consortium (W3C).

[10] Chebotko, Artem., Lu, Shiyong., Hasan, Jamil, M., Fotouhi, Farshad. (2006). Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns, Wayne State University, Technical Report TR-DB-052006-CLJF.

[11] Antal, Marius., Anechitei, Daniel., Alexandru Ioan Cuza, "SQL2SPARQL", University, Faculty of Computer Science, 16, General Berthelot St., 700483, Iaºi.

[12] Chebotko, Artem., Lu, Shiyong., Fotouhi, Farshad. (2007). Semantics Preserving SPARQL-to-SQL Query Translation, Wayne State University, Technical Report TR-DB-112007-CLF.

[13] Marcelo, Arenas., Gutierrez, Claudio et., Perez Jorge, (2010). On the Semantics of SPARQL, Springer Berlin Heidelberg, 2010.