International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 10, October-2015

# SQL to SPARQL Mapping for RDF querying based on a new Efficient Schema Conversion Technique

Larbi Alaoui
International University of Rabat,
Sala Al Jadida, Morocco

Ahmed Abatal, Khadija Alaoui, Mohamed Bahaj, Ilias Cherti
University Hassan 1st,
Faculty of Science and Technologies,
Settat, Morocco

***Abstract*— In this paper we give an algorithm for querying RDF data using SQL without conversion of RDF instances. This algorithm translates an SQL query into an equivalent SPARQL query that is to be directly executed on the RDF data and allows it for SQL users to efficiently and easily query the RDF data. The SQL queries are formulated based on the converted relational database schema that the algorithm builds from the RDF one. In this algorithm not only simple SQL queries are considered but also complex ones such as those with UNION, INTERSECT or EXCEPT expressions.**

***Keywords*— *RDB, RDF, SQL, SPARQL, Query translation***

## I. INTRODUCTION

Our aim in this paper is to facilitate querying of RDF data for SQL users by providing a framework for translating RDF schemas into relational database (RDB) schemas and an algorithm for translating SQL queries into SPARQL queries based on the proposed framework.

RDF (Resource Description Framework) which was standardized by the W3C [6] is a language for describing the semantics of data that allows sharing of its meaning between different applications. RDF provides a powerful data model based on representing data in RDF graphs that can be queried using SPARQL. SPARQL (SPARQL Protocol and RDF Query Language) was proposed and standardized by W3C [7] as a query language for RDF.

Because of the well established techniques of relational database (RDB) systems, storing and querying RDF using relational databases techniques is of great importance for multiple RDB applications which makes it an attractive research topic in the world of information retrieval. For example, different research works have been made for translating RDB into RDF [8-9].

It is however to be noticed that there is still a lot of research work to be done for translating RDF to RDB for exploring RDF data.

The main existing works with regards to the translation from RDF into RDB are those of Rachapalli & al. [1] and Ramanujam & al. [5]. In [1] Rachapalli & al. proposed a Framework that stores RDF into RDBMS using a vertically partitioning storage technique [2] where a table is created for each predicate and the table contains a subject-object as attributes (i.e. name (subject, object)). This framework is based on providing a relational model that contains therefore many tables. It therefore makes it inadequate for easily querying the resulting RDB tables using SQL since it involves adding many join conditions among these tables and makes the conversion from SQL into SPARQL a complex one. Also Ramanujam & al. [4] presented a tool for visualizing RDF into a virtual RDB that contains a module for translating SQL to SPARQL. His work focus in converting a SQL query with data aggregation abilities like GROUP BY and ORDER BY clauses without any details on how the conversion is made. Furthermore, the work in [4] does not consider some important SQL constructs such as UNION, INTERSECT and EXCEPT.

Our aim in this paper is to tackle the problems and the short outcomes of the aforementioned existing works. In this sense we propose a new solution for RDB users to easily extract information directly from RDF data by providing them with an associated RDB schema we carefully extract from the RDF schema and without a translation of the RDF data instances. Users can simply formulate their queries using the RDB query language SQL, and our newly developed algorithm based on the extracted relational schema will translate them into SPARQL ones that can therefore be executed directly on the RDF data. Contrary to the existing works our strategy for schema translation uses a simple structure for the schema modeling and therefore for the SQL to SPARQL translation. The RDF to RDB schema mapping with its simplicity allows us to treat not only simple queries but also those queries with more complex constructs that are relevant for users and that were not considered before such as UNION, INTERSECT and EXCEPT.

The remainder of this paper is organized as follows. In section II we give our schema mapping model. Section III presents our algorithm for SQL to SPARQL query conversion based on the aforementioned schema mapping model. Section IV gives a summary and open perspectives of our work.

## II. SCHEMA MAPPING

In this section we propose a new framework for modeling RDF data using relational database schemas.

In our contribution, we propose a new alternative for RDB modeling of RDF data without storing the data in the newly defined RDB tables and give a translation method of SQL queries into SPARQL equivalent ones which can therefore be

made on the real RDF data instances. First property tables are used to group subjects with similar predicates together. Every table contains a subject as attribute and a set of predicates attributes. In difference with the work done in [1] we don't use a table for each predicate. This allows us to come up with a solution with largely less tables and a consistent modeling of the data. The solution therefore reduces the join conditions and yields an efficient logical relational schema that facilitates querying the schema for users. In difference with the work done in [4-5] our solution come up with a consistent schema modeling that allows us to efficiently issue not only simple SQL queries but also complex ones such as nested queries.

RDF is based on modeling data in form of triplets where each triplet is constituted of a subject, a predicate and an object. An example illustrating such triplets is given in Figure 2.

For a given RDF model, our schema-mapping algorithm traverses all triplets of the Abox set and adds for each one its associated type and predicate to a hash map.

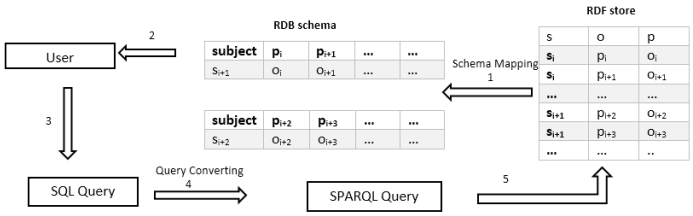


Fig 1: Mapping algorithm steps

The algorithm is the following one:

| *Schema-Mapping()* Algorithm - *Part 1* |
|---|
| Input: AB a set of Abox<br>Output: A hash Map, P (relation name, predicate name) |

```
1.  P ← θ; { The map P is initially empty }
2.    for i ← 1 to AB.size do
3.      T← AB[i] { Retrieve a triple T from the AB}
4.      r ←T.type { Retrieve the type r from the triple T }
5.      v ←T.predicate
6.       { Retrieve the predicate p from the triple T }
        notFound ← true
7.      j ← 1
8.      while ((notFound) AND (j<=P.size))  do
9.        if ( (r==P(j).getRelation()
10.         AND v ==P(j).getPredicate())
11.         then
12.           notFound ←false;
13.       end if
14.     end while
15.     if (notFound) then
16.       P.put (r,v ) { Add predicate v, and relation r to P }
17.     end if
18.   end for
19. Return P
```

The Hash map P returned by the *Schema-Mapping()* Algorithm will be used to extract relation names and their associated attributes names. For each key r of P we create a relation R and the attributes of R are simply the predicates v that are associated with r in P. An additional attribute *SUBJECT* is also added to the list of attributes of R to represent subjects of the relation r. The values of this attribute will then be the subject values of the RDF triplets of r. This transformation step of *Schema-Mapping()* is therefore as follows

| Algorithm *"Schema-Mapping()" - Part 2* | |
|---|---|
| Input: The hash Map P of Part-1<br>Output: Relational schema | |
| | For each key r of P<br>Create a relation *R* with an attribute *SUBJECT*<br>For each value v associated with r in P add an attribute v to R |

For the RDF example of Figure 2 we get the following relational tables "AUTHOR", "PUBLISHER", and "BOOK":

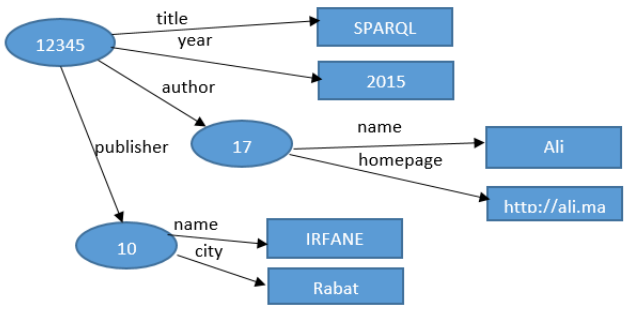


Fig 2: example for RDF graph

- AUTHOR = {subject, name, homepage}
- BOOK= {subject, title, year, author, publisher}
- PUBLISHER = {subject, name, city}

This example will also be considered in the next section to illustrate our SQL to SPARQL conversion results that use the relational schema we built from the RDF data.

## III. QUERY CONVERSION

In this section, we give a list of algorithms for mapping SQL queries into SPARQL ones. The resulted SPARQL queries can therefore be executed directly on RDF data. The conversion algorithms take into consideration the fact that there are equivalent types of the tables used in the SQL queries. They are therefore suitable for the use with the relational schema we extracted from the RDF one in the previous section. For the different section composing an SQL query, we give associated algorithms. More precisely, two algorithms *ConvSelectSql()* and *ConvWhereSql()* are separately given to yield the results of the SELECT part and of the WHERE part respectively, and a *combine()* algorithm is used for the combinations involved in the SQL query.

Table 4 gives an example of SQL queries for the relational scheme S and their equivalent SPARQL queries for the RDF schema that is obtained from our algorithm.

| Query Description | SQL Query | SPARQL Query |
|---|---|---|
| Author name of SPARQL Book | SELECT author.name<br>FROM author, book<br>WHERE<br>author.subject=book.author<br>AND<br>Book.title=”SPARQL” | SELECT ?o1<br>WHERE<br>{<br>?s1 name ?o1 .<br>?s2 author ?s1.<br>?s2 author ?o2.<br>?s2 title ?o3<br>FILTER(?o3=”SPARQL”)<br>} |
| Select titles of all books | SELECT book.title<br>From book | SELECT ?o<br>WHERE {<br>?s title ?o } |
| Names authors and publishers | SELECT author.name<br>FROM author<br>UNION<br>SELECT publisher.name<br>FROM publisher | SELECT ?o1<br>WHERE<br>{<br>? s1 name ?o1<br>}<br>UNION<br>{<br>? s1 name ?o1<br>} |

Table 4. Illustration of SQL query converted to SPARQL

### A. *Conversion algorithms*

The algorithm Query Converted is used to convert a input SQL query into an equivalent output SPARQL query, firstly the algorithm call *ConvSelectSql()*, take as an input SQL SELECT Clause, and return a SELECT SPARQL Clause, and a list of triple patterns names TP, for generating a triple pattern, $?s_i$ $r_i$ $?o_i$ for every relation ri, because we have attributes names in SQL SELECT Clause, to allows the next algorithm *ConWhereSql()* to index the list TP.

**ConvSelectSql() Algorithm**

Input: The list A of attributes of an SQL-Select query
Output: SPARQL Select, and triple patterns TP

```
1   select =""
2   TP ← empty set;
3   for i ← 1 to A.size do
4     r ←A{i}.relation { Retrieve the relation r from attributes A}
5     p ← A{i}.attribute { Retrieve the attribute p from A }
6     if p !='subject' then
7        select += "o" + i
8        tp←{ ?si p oi } { A triple pattern is constructed }
9        TP.put (r, tp )
10    else
11       select += "s " + i
12    end if
13  end for
14  Return select, TP
```

The algorithm ConvWhereSql() generates a correspondent SPARQL WHERE clause by evaluating the SQL WHERE clause, and takes as input: the join conditions JC (each condition has the form: Attribute Operator Attribute), the Boolean expressions BE (each condition has the form: Attribute Operator Value), the triple patterns generated by ConvSelectSql*()* algorithm, and the list of predicates from the Hash map outputted by the *Schema-Mapping()* algorithm.

The SPARQL WHERE clause is obtained by converting join conditions in a given SQL query into an equivalent triple patterns (lines 6-16) and by making a filtering using the boolean conditions (lines 24-30). In lines 18-22 it verifies that no triple pattern needed in the filtering with join condition with objects does miss.

**ConvWhereSql() Algorithm**

Input: JC: Join conditions, BE: Boolean expressions, TP (Relation name→ triple pattern)
Output: WHERE clause

```
1.  where = ""
2.  if(JC.isEmpty() AND BE.isEmpty()) then
3.    for each tp from TP do
4.      where += "?" + tp.subject + " " + tp.predicate +" ?"
5.                                                  + tp.object
6.    end for
7.  Else
8.    if (! JC.isEmpty ) then
9.          for each p from JC do
10.           p1 =p.LeftOperand ; p2 =p.RightOperand ;
11.           tp1 =TP.get(P1.relation)
12.           tp2 =TP.get(P2.relation)
13.           where += "?" + tp1.subject+" " + tp1.predicat +
14.                                              "?"+tp1.object
15.           if P1.relation = P2.attribut then
16.              where+= "?" + tp1.subject + " "+ P1.attribut + " ?"
17.                      + tp1.object +  "." + " ?" + tp2.subject +
18.                       " " + P2.attribut + " ?" + tp1.object + ". "
19.           end if
20.         End for
21.       {if we have a Boolean conditions }
22.       if (! BE.isEmpty ) then
23.         for each e from BE do
24.           p=e.LeftOperand;
25.           tp =TP.get(p.relation)
26.           where += "?" +  tp.subject + " " + tp.predicat +
27.                                              "?" + tp.object
28.         end for
29.         {for adding FILTER }
30.         for each e from BE do
31.           P1=e.LeftOperand;
32.           p2=e.RightOperand;
33.           tp =TP.get(p1.relation)
34.           where += "FILTER("+ tp.object+"  "
35.                                  + e.operator + " " + p2
36.         end for
37.       end if
38.       Return where
```

The core of query converting takes for its input an SQL query in a string format and the hash map P from the SCHEMA-MAPPING() algorithm, and gives a SPARQL query in a string format. The SQL query string is parsed to extract the clauses SELECT, WHERE-A (WHERE with join condition) and WHERE-B (WHERE with Boolean expressions). If the WHERE clause does not contain any join condition or Boolean expressions, then we set null for the value of the clause. This algorithm uses the previously given algorithms ConvSelectSql(), ConvWhereSql () to convert each considered clause

In the case the SQL query has an SQL UNION, EXCEPT or INTERSECT construct, it is simply considered as two composed SQL queries and the QueryConvert() algorithm first converts each one of these queries before combining the conversion results using the *combine()*-method given below in order to yield the final SPARQL query.

*QueryConvert()* Algorithm

Input: SQL query SQ, and hash map P
Output: SPARQL SPQ

```
1.  SPQ ="" { initialize SPARQL query}
2.  tnodes=analyze(SQ) {cut SQL query to obtain clauses }
3.  SQselect= tnodes.getSelectClause()
4.  SQwhere-A= tnodes.getWhereJC()
5.  SQwhere-B= tnodes.getWhereBE()
6.  SPQselect="SELECT "
7.  SPQwhere="WHERE { "
8.  TP ← θ;
9.  TP ←ConvSelectSql(SQselect).getTP()
10. SPQselect=ConvSelectSql(SQselect)
11. SPQwhere +=
12.     ConvWhereSql(SQwhere-A, SQwhere-B,TP,P)
13. SPQ= SPQselect+ SPQwhere+''}''
14. if tnodes.type!= null then
15.     q1=tnodes.leftSubSQL()
16.     q2=tnodes.RightSubSQL()
17.     SPQ1=queryConvert(q1)
18.     SPQ2=queryConvert(q2)
19.     SPQ=combine(SPQ1, SPQ2, tnodes.type)
20. end if
21. Return SPQ
```

The method "*analyze( )*" cited in line 2 takes an SQL query as input and return a set of nodes by splitting the SQL input and extracting separately the attributes from the SELECT clause, the join conditions and the Boolean expressions from the WHERE clause. The methods getWhereJC() and getWhereBE() used in line 4-5 extract after this the associated join conditions and the boolean expressions.

If we have an SQL sub-queries joined by a UNION, INTERSECT or EXCEPT type then the associated SPARQL queries are grouped together by QueryConvert() algorithm in lines 14-20 by call combine()-method which takes as input the sub-queries and the merging type.

*Combine()* Algorithm

Input: SPARQL Query q1; SPARQL Query q2; type {type is either INTERSECT, EXCEPT or UNION}
Output: A SPARQL query SPQ

```
1.  SPQ =" " {A SPARQL query that is initially empty}
2.  sparqlSelect= q1.getSelectClause()
3.  sparqlWhere=" { " ;
4.  sparqlWhere1=" { "+q1. getSparqlWhere()+" }";
5.   sparqlWhere2 = " { "+q2. getSparqlWhere ()+" }";
6.  sparqlWhere +=SpWhere1+type+SpWhere2+" }"
7.  SPQ+=SpSelect+SpWhere
8.  return SPQ
```

## B. IMPLEMENTATION

Our algorithm was implemented using the Java language. To test the SPARQL queries converted by our algorithm from SQL ones we used Jena's [3] ARQ module which is a Java-based project.
The following screen shots (Fig. 3-5) present some examples of the conversion results using our algorithm.

The SQL query considered in Fig 3 is the following one:

```
SELECT book.title
FROM book
```

The resulted SPARQL query is as follows:

```
SELECT ?o0
WHERE
{
    ?s0 ?title ?o0
}
```

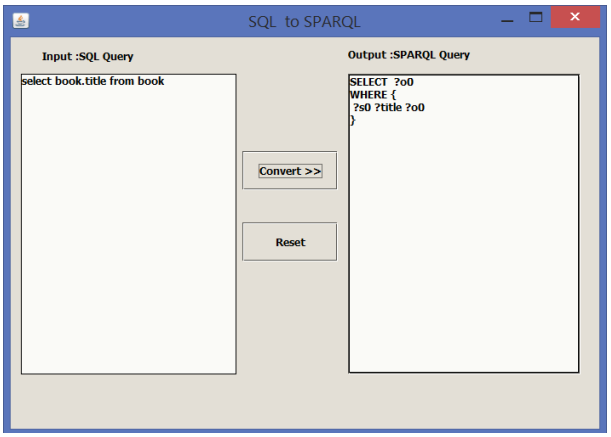


Fig 3: simple select Query

Fig 4 shows an example of the SQL query with join conditions and Boolean expressions given bellow:

```
SELECT author.name,book.title
FROM author book
WHERE author.subject =book.author
AND book.title=" SPARSL"
```

The equivalent SPARQL query outputted is:

```
SELECT ?o0
WHERE {
    ?s0 ?name ?o0.
    ?s1 ?title ?o1.
    ?s1 ?author ?s0.
    ?s1 ?author ?o2
    FILTER(?o1=="sparql" )
}
```

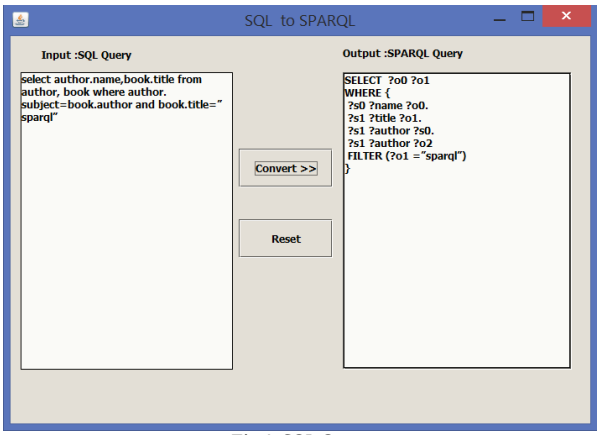


**Fig 4:** SQL Query

Fig 5 shows an example of the following SQL query containing a UNION construct:

```
SELECT author.name,
FROM author
UNION
SELECT publisher.name,
FROM publisher
```

Its associated SPARQL query obtained by the algorithm is:

```
SELECT ?o0
WHERE
{
    {
      ?s0 ?name ?o0
    }
    UNION
    {
      ?s1 ?name ?o0
    }
}
```

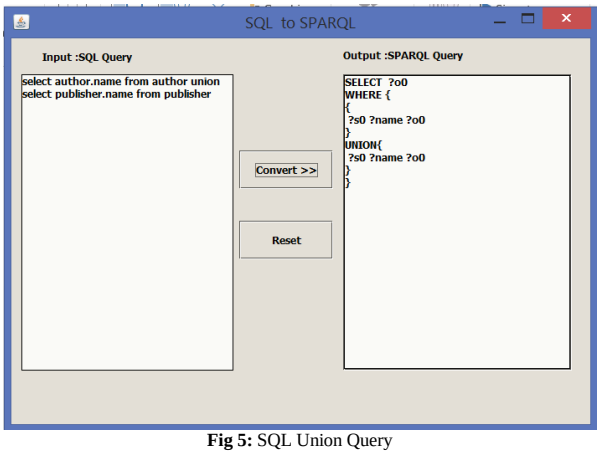


**Fig 5:** SQL Union Query

## V CONCLUSION

Resource Description Framework has been standardized by the W3C as the language of the semantic web to reflect the semantics of the data being exchanged on the web. It comes with an emerging data format that makes it possible to share the meaning of data between various applications. However because of the dominance of relational database systems and associated tools that are still based on SQL for handling data there is an increasing need for tools to help SQL users to query RDF data. In this perspective we proposed in this paper an approach for querying RDF data using SQL. The technique we used is based on modeling RDF data by a suitable relational schema that makes it possible for users to query RDF data with SQL without any instance translation into relational tables. Based on the extracted relational schema our approach converts users SQL queries into equivalent SPARQL queries to be executed on the RDF data. This is done in efficient way since the proposed modeling technique insures a consistent representation of all information in RDF data that avoids redundancy and comes up with a minimal set of representation tables in the extracted schema.

Because of this concise modeling technique we aim to further use it in the future for integration with existing relational database systems for purposes related to RDF data storage and manipulating. This will open a new era for existing relational systems to be open for extensions to the world of semantic web.

## REFERENCES

[1] Jyothsna Rachapalli, Vaibhav Khadilkar, Murat Kantarcioglu and Bhavani Thuraisingham, "RETRO: A Framework for Semantics Preserving SQL-to-SPARQL Translation", EvoDyn Workshop, 2011.

[2] Bajda-Pawlikowski, K.: Querying RDF data stored in DBMS: SPARQL to SQL Conversion. Technical Report TR-1409, Yale Computer Science Department, USA Jena Framework, https://jena.apache.org.

[3] Sunitha Ramanujam , Anubha Gupta , Latifur Khan , Steven Seida , Bhavani Thuraisingham, "R2D: A Bridge between the Semantic Web and Relational Visualization Tools", Proceedings of the 2009 IEEE International Conference on Semantic Computing, p.303-316, September 14-16, 2009.

[4] Sunitha Ramanujam, Anubha Gupta, Latifur Khan , Steven Seida, Bha vani M. Thuraisingham "R2D: Extracting relational structure from RDF stores" In Proc. of ACM/IEEE International Conference on Web Intelligence, Page 361-366, September, 2009, Milan, Italy

[5] W3C, Resource description framework (RDF): concepts and abstract syntax, in: G. Klyne, J.J. Carroll, B. McBride (Eds.), W3C Recommendation[S], 10 February2004. http://www.w3.org/TR/2004/REC-rdf-concepts- 20040210/

[6] W3C, SPARQL query language for RDF,in: E. Prud'hommeaux, A. Seaborne (Eds.), W3C Recommendation[S], 15 January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql -query-20080115/

[7] S. Das, S. Sundara, and R. Cyganiak. 'R2RML: RDB to RDF mapping language', Sept. 2012. URL http://www.w3.org/TR/2012/REC-r2rml/20120927/

[8] J. Grabis, M. Kirikova, 'Advanced RDB-to-RDF/OWL Mapping Facilities in RDB2OWL',10th International Conference, BIR 2011, Riga, Latvia, October 6-8, 2011.pp 142-157