Available online at www.sciencedirect.com

Journal of Applied Research and Technology

Journal of Applied Research and Technology 15 (2017) 504–512

www.jart.ccadet.unam.mx

Review

Semantics preserving SQL-to-SPARQL query translation for Nested Right and Left Outer Join

Nassima Soussi *, Mohamed Bahaj

Hassan 1st University, Faculty of Science and Technologies/Department of Mathematics & Computer Science, Settat, Morocco

Abstract

Despite the emergence of semantic web due to its high performance in managing a large amount of data through semantic filters, the relational databases are still the most used. Therefore establishing a connection between both heterogeneous systems becomes a relevant need so as to bridge the gap between them. Regarding the query mapping from relational world to semantic world (SQL-to-SPARQL), some solutions have been developed to realize this transformation but unfortunately, all existing approaches do not consider the semantic correspondence between Left/Right Outer Join command(s) and Optional pattern(s). This weakness has motivated us to investigate in this direction and suggest an efficient solution to deal with this problem by proposing, to the best of our knowledge, the first query mapping algorithm ensuring the semantic transformation of simple/nested Left/Right Outer Join command(s) in SQL queries to a simple/nested Optional pattern(s) in SPARQL equivalent ones without physical transformation of data.

Keywords: SQL-to-SPARQL; Query transformation; Nested Left Outer Join; Nested Right Outer Join; Nested optional pattern

1. Introduction

The majority of web information is stored currently in relational databases since they were dominant for the past several decades due to their simplicity and performance in managing data. However, the amount of information is growing day by day exposing the web faced inevitable problems owing of the low competence of this traditional system to manage intelligently this big amount of data as well as the absence of logical reasoning in query answering. In order to overcome the previous gaps and enable not only to users but also to machines to find, share, and combine information more easily, the W3C has given birth to semantic web aiming to exploit the full web potential and permit machines to intelligently access different data sources. This semantic data is represented via a standard model called RDF (Resource Description Framework) (Vertan & Merkmale, 2004), characterized by a simple and powerful structure of triples (Subject-Predicate-Object); the subject represents the resource and the predicate represents the relationship between the subject and object. This structure makes semantic data very similar to human language (Subject-Verb-Object).

Since most of RDBMS users are unfamiliar with semantic web philosophy, principles and technologies, it becomes a relevant need to establish for them a suitable bridge to query RDF data. Therefore, some researchers have been made to achieve this goal and ensure a query mapping from relational world to semantic world (SQL-to-SPARQL), but unfortunately, all these approaches have common weaknesses in the consideration of the equivalence between Left/Right Outer Join command(s) and Optional pattern(s) in the query mapping translation. This problem has motivated us to operate in this topic so as to remedy this gap and establish, to the best of our knowledge, the first algorithm converting SQL queries containing Left/Right Outer Join command(s) (simple and nested form) to SPARQL ones with Optional pattern(s) (simple and nested). Our system is very helpful for organizations working entirely with RDBMS and aiming to interact with semantic databases without spending so much in the migration of their system and training their users in the semantic technologies new for them, in addition, this solution is beneficial in terms of complexity and execution time of queries by replacing the heavy and costly SQL joins with SPARQL Optional clauses. Another advantage of our solution is that it facilitates connection and interoperability between relational

* Corresponding author.
E-mail address: nassima.soussi@gmail.com (N. Soussi).
Peer Review under the responsibility of Universidad Nacional Autónoma de México.

http://dx.doi.org/10.1016/j.jart.2017.05.001

databases and RDF stores with out physical transformation of data.

The remainder of this paper is structured as follows: Section 2 analyses some existing approaches to the current topic. Section 3 introduces the context of this work by describing the syntax of each query language via its own metamodel and discussing the similarity and discordances between them. Section 4 describes by examples a different mechanism encapsulated in our strategy. Our contribution is presented in Section 5 that exposes our query mapping algorithm allowing the semantic transformation of Left Outer Join command (in its simple and nested format) to SPARQL equivalent Optional pattern. In Section 6 we expose the developed java application implementing our solution. Finally, Section 7 concludes this work and suggests some future extensions of this topic.

2. Related works

In order to establish an efficient connection between the semantic web and relational world (considered as the most used database management system), more precisely, to ensure the query interoperability between SQL and SPARQL languages, several researches have been developed regarding SPARQL-to-SQL direction, we quote as example:(Chebotko, Lu, & Fotouhi, 2009; Rodriguez-Muro & Rezk, 2015) and (Atre, 2015); all these papers have considered the semantic equivalence between Left Outer Join and optional pattern but they did not treat the Right Outer Join. On the other hand, there are not so much works related to the reverse direction (SQL-to-SPARQL represented the subject of the current work), and the few existing approaches have several weaknesses as presented subsequently.

RETRO method (Rachapalli, Khadilkar, Kantarcioglu, & Thuraisingham, 2009) contribute in the interoperability between RDF Stores and RDBMS by firstly deriving a relational schema from the RDF store, and secondly providing a means to translate an SQL query to a semantically equivalent SPARQL query. RETRO deals with schema mapping in addition to query mapping and chooses not to physically transform the data. However, the authors consider just a basic query in its transformation process and they have not proposed an implementation of set of experiments.

SQL2SPARQL solution ("SQL2SPARQL", in press.) aims also to transform SQL queries into SPARQL ones using some dynamic mappings and transformation rules based on the combination of ideas already presented in other works. In this approach, only classic and basic queries are supported and they did not describe their transformation method in a detailed way via an algorithm, in addition, they have just assembled and combined the tools of some existing works in order to carry out this paper and they have not added anything new. However, they created a java application to demonstrate the effectiveness of their work.

R2D method (Ramanujam, Gupta, Khan, Seida, & Thuraisingham, 2009) operates in the same mapping direction; it proposes a mechanism which enables reusability of relational tools on RDF data including SQL-to-SPARQL translation; R2Dsystem converts SQL queries (with pattern matching, aggregation and blank nodes) into the SPARQL equivalent ones. This translation is presented with algorithmic details validated via an implementation of set of experiments.

The authors in (Alaoui, Abatal, Alaoui, Bahaj, & Cherti, 2015) propose an algorithm for querying RDF data using SQL language by translating SQL queries (simple and complex ones containing UNION, INTERSECT or EXCEPT expressions) into an equivalent SPARQL queries based on modeling RDF data by a relational schema. They have concluded this work by presenting screenshots of the implementation of experiments.

All the previous approaches have the same and common weakness that they do not consider the equivalence between Right and Left Outer Join and Optional pattern(s) in their transformation mechanism of rewriting SQL queries into their semantically equivalent SPARQL queries. The first work established in this direction that highlighted this equivalence is presented in (Ramanujam, Gupta, Khan, Seida, & Thuraisingham, 2009) by indicating the ability to optionally retrieve reification data, when present, through joins; nevertheless, this approach does not propose a clear transformation rules or proposing a detailed algorithm describing the different steps to ensure this equivalence. In addition, all quoted approaches do not consider a nested form of Left/Right Outer Join and Optional.

3. Query languages description

In this section, we describe the query languages treated in our work in order to present the concept of each one via its own metamodel: the famous relational query language designed for managing data in a RDBMS (SQL) and the semantic query language dedicated to query RDF stores (SPARQL).

3.1. SQL: structured query language

SQL is a standard query language for operating relational databases by providing the ability to search, add, modify or delete data. This language is created in 1974, standardized since 1986, and currently, it is recognized by the vast majority of RDBMS users.

SQL queries can have numerous possible types (CREATE, INSERT, SELECT, UPDATE, DELETE and DROP), but in the current study, we are only interested by SELECT query containing Left and Right Outer Join command(s). As illustrated in Figure 1, SQL SELECT query ("SQL Grammar", in press) consists of six clauses: *SelectClause*,*FromClause*, *WhereClause*, *GroupByClause, OrderByClause* and *HavingClause*; the two first clauses are mandatory in SELECT SQL queries.

SELECT clauseis composed ofa set of properties that will be appeared in the result, where as FROM clause can contain one or more table(s)when the query's type is simple (**SELECT** varList **FROM** table(s)_name(s) . . .), or squarely an SQL Join clause when the query's type is complex; this second case aims to combine data from multiple tables efficiently in order to exploit

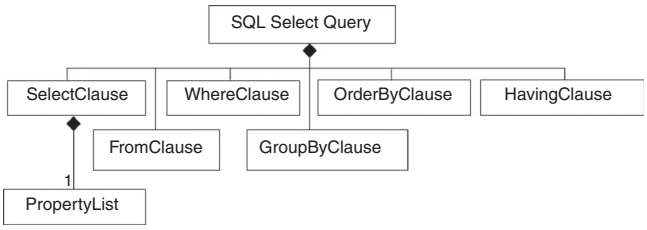

Fig. 1. Metamodel of SQL SELECT Clause.

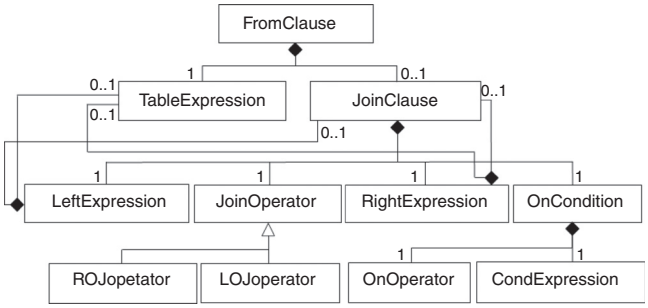

Fig. 2. Metamodel of SQL FROM Clause.

the power of relational databases. SQL language has several types of Join: (1) *InnerJoin* returns all rows when there is a match in both tables, (2) *LeftOuterJoin (LOJ)* returns all rows from the left table and the matched rows from the right table, (3) *RightOuterJoin (ROJ)* returns all rows from the right table and the matched rows from the left table and (4) *FullOuterJoin Return* all rows when there is a match in one of the tables. In this paper, we focus on Left and Right Outer Join described in Figure 2 representing the main subject of our study.

The Left and Right Outer Join are composed of four mandatory statements: Left Expression, Join Operator (LOJ or ROJ), Right Expression and On Condition representing the join criteria for matching rows that will be returned in the result. The left and right expressions can be simple (table) or complex (joining query) depending on SQL query type (simple or nested Left/Right Outer Join query).

3.2. SPARQL: RDF query language

SPARQL (**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage) [a], as its name indicates, is both a protocol and query language that is able to manipulate data stored in RDF format. It is considered as a standard and one of the key technologies of the semantic web. In addition, SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions which indicates that this famous language is based on the graph pattern concept in the selection of data (RDF triples) from RDF source. In order to carry out our mapping approach and establish an efficient algorithm, we have defined a metamodel of the target language describing its concrete syntax.

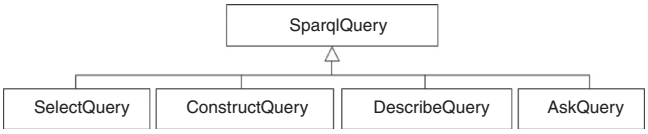

Fig. 3. Metamodel of SPARQL Query Types.

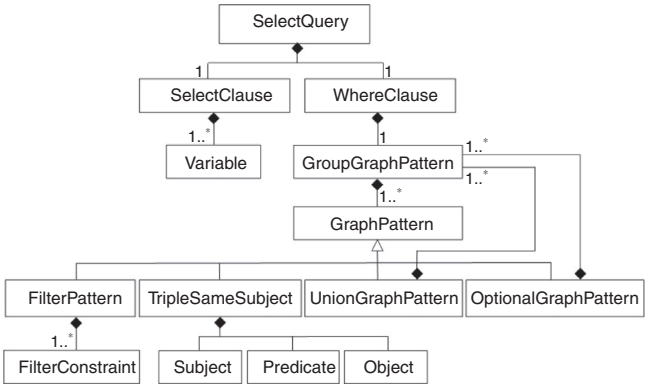

Fig. 4. Metamodel of SPARQL SELECT Query.

SPARQL query can have a different possible type as described in Figure 3 (Select, Construct, Describe and Ask); in this study, we are only interested by SELECT query as shown in Figure 4 illustrating a metamodel of this later based on the SPARQL grammar (Harris, Seaborne, & Prud'hommeaux, 2013).

SELECT queryconsists of two main statements: (1) *SelectClause* identifies the variables to appear in the results, and (2)*WhereClause* which consists, in its turn, of *GroupGraphPattern* that identifies a set of *GraphPattern* represented in multiple forms. We quote:

- *FilterPattern*: used to filter a set of objects using a various criteria and requirements. The filter expressions can be combined through the logical operations so as to form more complex FILTER constraints.
- *TripleSameSubject*: includes a subject and associated properties.
- *UnionGraphPattern*: union of patterns.
- *OptionalGraphPattern*: optional patterns.

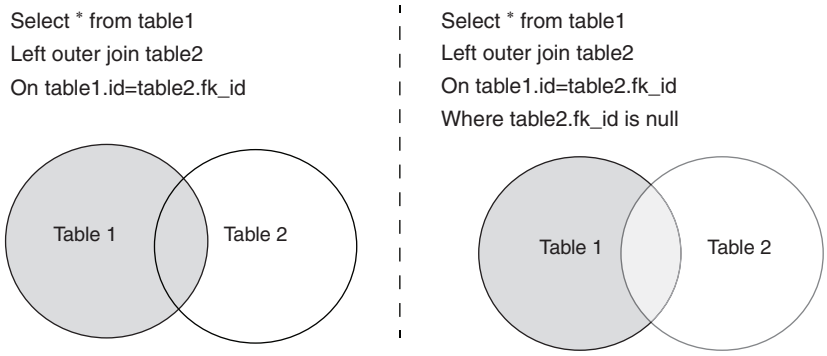

Fig. 5. Left Outer Join Description.

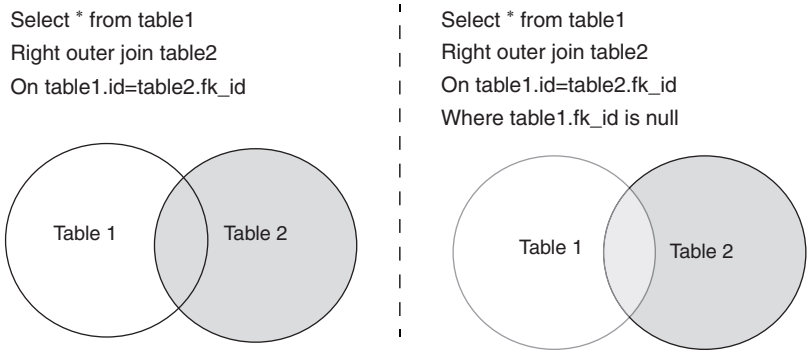

Fig. 6. Right Outer Join Description.

3.3. SQL vs. SPARQL

At first blush, SQL and SPARQL seems similar since they offer to users the possibility of accessing, creating, combining, and consuming structured data, but in reality, the direct comparison between them is rather difficult (Kumar, Kumar, & Kumar, 2011; Singh & Jain, 2014). In fact, SQL operates on relational data represented as tables and SPARQL operates on semantic data in RDF stores represented as triples (subject, predicate and object).

According to the metamodels elaborated above which describe the concepts of each language, numerous commonalities exist between them:

- The aggregate functions using with GROUP BY operator,
- Ordering query's results via the ORDER BY operator,
- Logic operators,
- Optional data represented with Left Outer Join command(s) in SQL and Optional pattern(s) in SPARQL.

Otherwise, there is a large difference between this both languages on several points:

- Data source structure: SQL operates on tables whereas SPARQL operates on triples.
- Syntax (look SQL/SPARQL metamodels illustrated above).
- Query types: we quote as example that SPARQL language supports ASK queries (returns a Boolean result indicating whether a query pattern matches) which are unknown for SQL language.

4. Strategy overview

4.1. Right Outer Join vs. Left Outer Join

In relational terms, SQL language has Left Outer Join key word used when we join two relations and we need to preserve all tuples of the first one in the result; this later can be NULL in the right side when there is no match between these two relations. This command is used as presented in Figure 5.

On the other hand, Right Outer Join keyword as presented in Figure 6, is used to join two relations in order to preserve all tuples of the second one (right table) in the result; we can have NULL tuples in the left side when there is no match between these two relations.

Similarly, semantic web world offers via its query language SPARQL the Optional pattern that aims to match a graph patterns, but if the optional match fails, the whole query does not fail, and in this case, a NULL value will be returned for unbound (no value) variables. Hence, we deduce that the basic equivalent semantic of Left Outer Join is Optional pattern. In addition, if we consider the conversion rule between LOJ and ROJ defined as *Table1* ***ROJ*** *Table2*<=>*Table2* ***LOJ*** *Table1*, we conclude that the principle of LOJ and ROJ in relational databases is equivalent to Optional pattern(s) in semantic world.

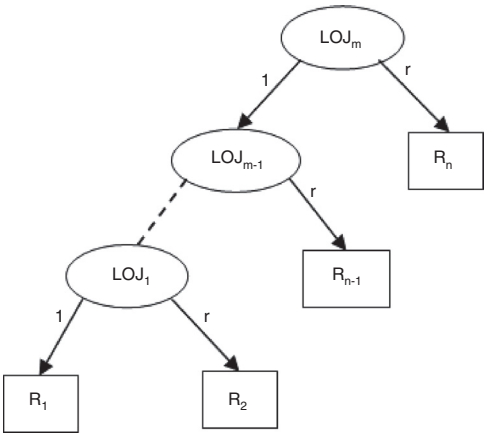

Fig. 7. General form of LOJ Binary Tree T.

4.2. Methodology description

Our strategy aims to enhance the SQL-to-SPARQL existing approaches by proposing the first efficient algorithm ensuring a semantic transformation, when exist, of simple/nested LOJ and ROJ command(s) in SQL queries to a simple/nested Optional pattern(s) in SPARQL equivalent queries. To achieve this goal, we have followed two steps:

- **Step 1:**Extracting the joining part from the input SQL query and we convert it to an equivalent binary tree in order to facilitate accessing its different components,
- **Step 2:**Join tree to Optional pattern(s): if the join's type is LOJ, we develop an algorithm for transforming LOJ command(s) (in its simple or nested form) represented as a tree to an equivalent Optional pattern(s), else if the join's type is ROJ, we exploit the conversion rule between LOJ and ROJ (described in the precedent paragraph) to transform the ROJ tree to a LOJ tree and use the previous algorithm to ensure this conversion in order to avoid reinventing the wheel.

Subsequently, we describe how our strategy for LOJ & ROJ-to-Optional works.

4.2.1. LOJ-to-optional

As discussed above, the first step in our translation approach is to parse the joining part of input SQL query to a binary tree T as presented in Figure 7, (in this case, we suppose that the joining part contains LOJ command(s)) in order to glance through it and treat its different nodes starting with the last left leaf so as to preserve the left associativity of Optional patterns semantics. We use R_i as a relation (table) of rank i with $i \in [1, \ldots, n]$ and n is the number of relations in the input SQL query Q_{SQL}. In addition, LOJ_j denotes Left Outer Join clause of rank j with $j \in [1, \ldots, m]$ and m is the number of LOJ commands in the whole Q_{SQL}; we talk about nested LOJ query if $m > 1$. In fact, indices i and j express the execution order of relations and LOJ respectively.

- Case 1: Simple Left Outer Join

This case is considered as the basic one aiming to transform each component (node) of LOJ tree (table1 LOJ table2 ON table1.key = table2.key) to an equivalent SPARQL pattern(s) using a set of semantic correspondence rules defined as follows:

- Rule 1: Join attribute in the ON condition of LOJ query is equivalent to SPARQL pattern's subject (gp1 Opt {gp2}).
- Rule 2: The names of SQL join relations are equivalent to SPARQL predicates.
- Rule 3: SQL SELECT attribute(s) concatenated with '?' is equivalent to SPARQL object(s) (excepting the attribute considered as subject).
- Rule4: The pattern conceived from the right relation corresponds perfectly to the Optional pattern.

- Case 2: Nested Left Outer Join

In order to transform nested LOJ designed as $(((R_1 \boldsymbol{LOJ} R_2) \boldsymbol{LOJ} R_3) \ldots \boldsymbol{LOJ} R_{n-1}) \boldsymbol{LOJ} R_n$ to nested Optional patterns defined as $gp_1OPT(gp_2OPT(gp_3OPT(\ldots gp_{n-1}OPT)gp_n))$, the input SQL query have to respect some semantic rules.

- Rule 1: Shared variables between join relations must be bound to the same values. We consider for example the SQL query presented subsequently; join relations are $R_1(a,b)$, $R_2(a,c)$ and $R_3(a,c,d)$. Hence, the shared variable that must be bound is c ($R_{res}.c = R_3.c$).
  SELECT $R_{res}.b$, $R_{res}.c$
  FROM(**SELECT** $R_1.a$, $R_1.b$, $R_2.c$
  FROM R_1 **LEFT OUTER JOIN** R_2
  ON $(R_1.a = R_2.a)$
) **AS**R_{res} **LEFT OUTER JOIN** R_3
  ON $(R_{res}.a = R_3.a$ **AND** $R_{res}.c = R_3.c)$
- Rule 2: Before the evaluation of a main LOJ clause, all containing LOJ clauses have succeeded. Hence, the attributes of their right relations must be NOT NULL. If we consider for example the SQL query presented subsequently; join relations are $R_1(a,b)$, $R_2(a,c)$ and $R_3(b,d)$.
  SELECT $R_{res}.b$, $R_{res}.c$, $R_3.d$
  FROM(**SELECT** $R_1.a$, $R_1.b$, $R_2.c$
  FROM R_1 **LEFT OUTER JOIN** R_2
  ON $(R_1.a = R_2.a)$
) **AS**R_{res} **LEFT OUTER JOIN** R_3
  ON $(R_{res}.b = R_3.b$ **AND** $R_{res}.c$ IS NOT NULL)
  Hence, $R_{res} = \prod R_1.a, R_1.b, R_2.c\ (R_1 = \bowtie R_1.a = R_2.a\ R_2)$ must succeed before evaluating the main join represented as $R_{result} = \prod R_{res}.b, R_{res}.c, R_3.d (R_{res} = R_{res}.b = \bowtie R_3.b \cap R_{res}.c$ IS NOT NULL $R_3)$.

After ensuring that the input SQL query has respected the previous rules, we use the same translation process for each LOJ command.

4.3. ROJ-to-LOJ

In order to avoid reinventing the wheel and developing another algorithm to transform ROJ part in SQL queries to an optional pattern(s), we have added a new functionality to our strategy aiming to convert a ROJ part represented as a tree to a

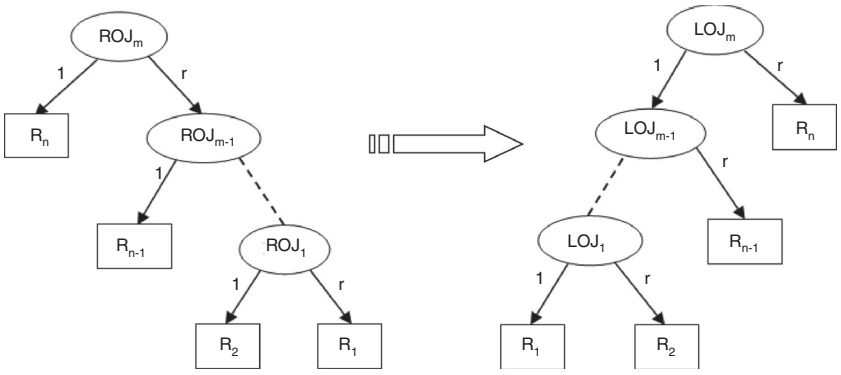

Fig. 8. ROJ to LOJ tree conversion.

LOJ tree as illustrated in Figure 8, and continue the process with the same algorithm.

5. Mapping description

In this section, we describe our main contribution by detailing all procedures used in our SQL-to-SPARQL algorithm allowing the semantic conversion of Left and Right Outer Join command(s) (in its simple and nested form) to SPARQL equivalent Optional pattern(s). Our algorithm is composed of five procedures: *QueryMapping, ParseROJ2LOJ*, *LOJ2Optional,CounterNbrLOJ* and *ConstructSPARQLSelectClause*.

5.1. Query mapping procedure

The main procedure *QueryMapping* takes as input an SQL query which contain Left or Right Outer Join command(s) (q_{in}) so as to return at the end a SPARQL equivalent query with Optional pattern(s). Firstly, we parse q_{in} to a binary tree in order to extract each component of SQL query separately, and then we parse also the FROM clause (containing LOJ or ROJ command(s)) to a binary tree named T; if the type of FROM clause tree is ROJ tree then we use the sub-procedure *ParseROJ2LOJ* in order to use the same treatment for both join types. Our procedure glances through the left side of a tree T till the end, starting with the root node (representing the main LOJ clause) so as to carry out the LOJ to Optional translation with *LOJ2Optional* sub-procedure when we reach the T tree's leaves (i = NbrLOJ), otherwise (if i ! = NbrLOJ) we store the right relation of each LOJ node and ON condition in rightRelStack and on CondRelStack stacks respectively for further processed. As long as the previous stacks is not empty, the conversion process continues in the WHILE loop via LOJ2Optional sub-procedure giving R_{res} that is concatenated in each iteration in order to form the SPARQL WHERE clause. Regarding SPARQL equivalent SELECT query is constructed through ConstructSparqlSelectClause sub-procedure. The cpt variable is incremented after each call of the sub-procedure LOJ2Optional in order to compute the number of OPTIONAL patterns; this variable is used later in the conception of the final query so as to close the braces of each OPTIONAL command.

Input: SQL LOJ query, q_{in}
Output: Equivalent SPARQL query, q_{out}
Begin
q_{out} = "", where = "WHERE{", cpt_opt = 0
Stack rightRelStack, onCondRelStack = NULL
ismainLOJ = False
Tree tree = parse(q_{in})
q_{in}^{SELECT} = tree.getSelectClause()
q_{in}^{FROM} = tree.getFromClause()
Tree T = parse(q_{in}^{FROM})
if (T.getType() = ROJtree) **then**
T = ParseROJ2LOJ(T)
end if
NbrLOJ = CounterNbrLOJ(T)
for $i \leftarrow 1$ **to**NbrLOJ**do**
if(i = 1) **then**
LOJ_i = root(T)
else
$LOJ_i = LOJ_{i-1}$.getLeftChild()
end if
if(i = NbrLOJ) **then**
ismainLOJ = True
end if
$R_R = LOJ_i$.getRightChild()
$R_L = LOJ_i$.getLeftChild()
ONcond = LOJ_i.getOnCondition()
if (R_L.isLOJQuery() = True) **then**
rightRelStack.push(R_R)
onCondRelStack.push(ONCond)
continue //go to the next iteration
else
R_{res} = LOJ2Optional(R_L,R_R,OnCond,ismainLOJ)
where + = Rres
cpt_opt++
end if
End for
ismainLOJ = False
While (rightRelStack.isEmpty = False) **do**
R = rightRelStack.pop()
OnCond = onCondRelStack.pop()
R_{res} = LOJ2Optional(R_{res},R, OnCond, ismainLOJ)
where + = R_{res}
cpt_opt++
End while
While (cpt ! = 0) **do**
where + = "}"
cpt = cpt −1
End while
select = ConstructSparqlSelectClause(q_{in}^{SELECT})
q_{out} = select + where + "}"
Return q_{out}
End Algorithm

5.2. ParseROJ2LOJ sub-procedure

The sub-procedure *ParseROJ2LOJ*takesas input a ROJ tree T_{ROJ} so as to glance through it and construct the LOJ equivalent tree T_{LOJ} by considering the correspondence rule (described previously) between this two join types (*T1* ***ROJ*** *T2* ⇔*T2****LOJ*** *T1*).

Input: A tree of Right Outer Join Query, TROJ
Output: A tree of Left Outer Join Query, TLOJ
Begin
R_l = "", R_r = "", n = getRootElement(TROJ)
while(n ! = NULL) **do**
R_l = n.getLeftChild()
R_r = n.getRightChild()
TLOJ = Tree("LOJ") //Initialization of TLOJ tree with the root element "LOJ"
TLOJ.InsertRightChild(Rl)
if(R_r = "ROJ") **then**
TLOJ.InsertLeftChild("LOJ")
$n = R_r$
else
TLOJ.InsertLeftChild(Rr)
n = NULL
end if
end while
returnTLOJ
End Algorithm

5.3. LOJ2 optional sub-procedure

The *LOJ2Optional* sub-procedure takes as input the different components of LOJ command: R_L(left relation), R_R(right relation), OnCond(LOJ condition) and ismainLOJ Boolean variable, in order to return at the end the semantic equivalent OPTIONAL pattern based on the semantic correspondence between LOJ and Optional.

Input: R_L, R_R, OnCond, ismainLOJ
Output: Optional patterns
Begin
Subject, $Predicate_L$,$Predicate_R$, $Object_L$,$Object_R$,
Result = ""
Subject = OnCond.getFirstCond().getCommunAttr()
if (ismainLOJ = True) **then**
$Predicate_L$ = ":" + R_L.getName()
$Object_L$ = "?" + R_L[1] //The second attribute
BGP_L = Subject + $Predicate_L$ + $Object_L$
Result = BGP_L
end if
$Predicate_R$ = ":" + R_R.getName()
$Object_R$ = "?" + R_R[1]
BGP_R = Subject + $Predicate_R$ + $Object_R$
Result + = "OPTIONAL {" + BGP_R
Return Result
End Algorithm

5.4. CounterNbrLOJ sub-procedure

The CounterNbrLOJ sub-procedure takes as input a LOJtree T_{LOJ} aiming to glance through it and count all nodes unless leaves in order to return at the end the number of LOJ nodes.

Input: A tree T_{LOJ}
Output: A number of LEFT OUTER JOIN, NbrLOJ
Begin
Node = "", NbrLOJ = 0
if (T ! = NULL) **then**
node = Root(T_{LOJ})
NbrLOJ = 1
while (node.hasChildren() = True) **do**
if(LeftChild(node)! = NULL AND LeftChild(node).isLeafNode() = false) **then**
NbrLOJ = NbrLOJ + 1
node = LeftChild(node)
end if
end while
end if
returnNbrLOJ
End Algorithm

5.5. ConstructSparqlSelectClause sub-procedure

The ConstructSparqlSelectClause sub-procedure takes as input a set of SQL SELECT attributes in order to glance through it and extract their name so as to construct the SPARQL equivalent SELECT clause which is returned at the end of this algorithm.

Input: SQL Select Clause, V
Output: SPARQL Select Clause
Begin
Select = "SELECT"
Fori ← 1 **to**V.size**do**
Select + = "?" + V[i].getAttributeName() + " "
End for
Return Select
End Algorithm

6. Implementation

In order to implement our approach, we have developed a java application (in order to be portable and easy to evolve). Our tool is characterized by its simplicity and efficiency to allowing RDB users to query semantic data with SQL query language. The experiments were carried out on the PC with 2.4 GHz Core i5 CPU and MS Windows Seven Titan.

We present below some examples of SQL queries supported by our system and its equivalent SPARQL queries. The first one treatthe conversion of a simple SQL query that join two relations *PhdStudent* and *Advisor* with Right Outer Join type to an equivalent SPARQL query with one Optional pattern (Fig. 9), and the second one illustrates the translation of a complex SQL query with nested Left Outer Join (two Joins) to an equivalent SPARQL query with two Optional patterns (Fig. 10).

7. Conclusion

We have contributed through this paper in the enhancement of SQL-to-SPARQL translation approaches since there are not so much works related to this later, and all existing ones do not treat the equivalence between Right/Left Outer Join command(s) (simple and nested) and Optional pattern(s) (simple and nested) in their transformation mechanism of rewriting SQL queries into their semantically equivalent SPARQL queries. To the best of our knowledge, there is no work to date that fills this gap.

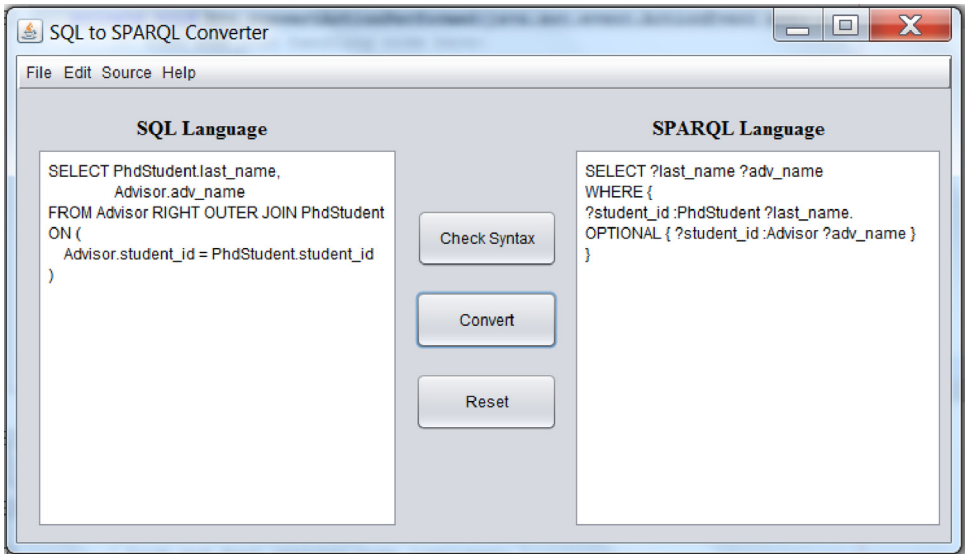

Fig. 9. Conversion example of SQL query with simple Right Outer Join.

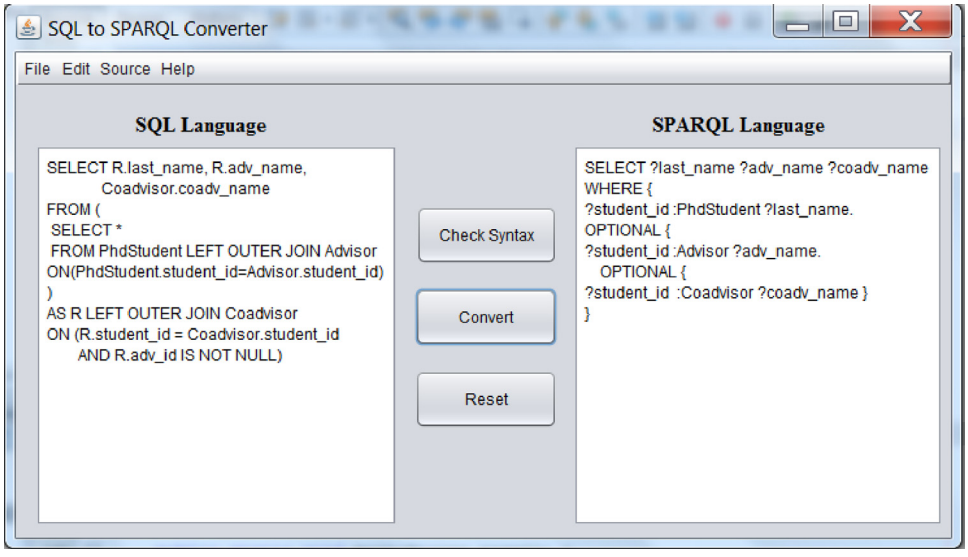

Fig. 10. Conversion example of SQL query with nested Left Outer Join.

One obvious extension of our research is to include the optimization strategies of SQL and SPARQL queries in our approach. Another promise about our future works is to integrate this conversion technique to RDBMS aiming to offer to relational users, an easy bridge and an open extension toward semantic world.

Conflict of interest

The authors have no conflicts of interest to declare.

References

Alaoui, L., Abatal, A., Alaoui, K., Bahaj, M., & Cherti, I. (2015, July). SQL to SPARQL mapping for RDF querying based on a new Efficient Schema Conversion Technique. *International Journal of Engineering Research and Technology*, *4*(10). IJERT.

Antal, M., & Anechitei, D. SQL2SPARQL [PDF document]. Retrieved from Lecture Notes Online Web Site: http://studylib.net/doc/7084652/sql2sparql.

Atre, M. (2015, May). Left bit right: For SPARQL join queries with OPTIONAL patterns (left-outer-joins). In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (pp. 1793–1808). ACM.

Chebotko, A., Lu, S., & Fotouhi, F. (2009). Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, *68*(10), 973–1000.

Harris, S., Seaborne, A., & Prud'hommeaux, E. (2013). *SPARQL 1.1 query language. W3C recommendation*. pp. 21. Retrieved from http://www.w3.org/TR/sparql11query/#sparqlGrammar

Rachapalli, J., Khadilkar, V., Kantarcioglu, M., & Thuraisingham, B. (2009). *RETRO: A framework for semantics preserving SQL-to-SPARQL translation*. 800 West Campbell Road, Richardson, TX 75080-3021, USA: The University of Texas at Dallas.

Ramanujam, S., Gupta, A., Khan, L., Seida, S., & Thuraisingham, B. (2009a). A relational wrapper for RDF reification. In *IFIP international conference on trust management* (pp. 196–214). Berlin, Heidelberg: Springer.

Ramanujam, S., Gupta, A., Khan, L., Seida, S., & Thuraisingham, B. (2009b). R2D: A bridge between the semantic web and relational visualization tools. In *IEEE international conference on semantic computing ICSC'09* (pp. 303–311).

Rodriguez-Muro, M., & Rezk, M. (2015). Efficient SPARQL-to-SQL with R2RML mappings. *Web Semantics: Science, Services and Agents on the World Wide Web*, *33*, 141–169.

Singh, M., & Jain, P. (2014). Time based query comparison of relational database and resource description framework (RDF). *International Journal of Advanced Research in Computer Science and Software Engineering*, *4*(12). ISSN: 2277128X.

SQL Grammar. Retrieved from http://www.h2database.com/html/grammar.html#select.

Vertan, C., & Merkmale, R. F. (2004). *Resource description framework (rdf)*.

Kumar, A. P., Kumar, A., & Kumar, V. N. (2011). A comprehensive comparative study of SPARQL and SQL. *International Journal of Computer Science and Information Technologies*, *2*(4), 1706–1710.