Semantic SPARQL query in a relational database based on ontology construction

Mohamed A.G. Hazber[1], Ruixuan Li[1+], Xiwu Gu[1], Guandong Xu[2], Yuhua Li[1,3]

[1]School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
moh_hazbar@yahoo.co.uk,
{rxli, guxiwu}@hust.edu.cn

[2]Advanced Analytics Institute,
Faculty of Engineering & IT
University of Technology Sydney
Sydney, Australia
guandong.xu@uts.edu.au

[3]State Key Laboratory of Software Engineering
Wuhan University
Wuhan, China
idcliyuhua@hust.edu.cn

Abstract— **Constructing an ontology from RDBs and its query through ontologies is a fundamental problem for the development of the semantic web. This paper proposes an approach to extract ontology directly from RDB in the form of OWL/RDF triples, to ensure its availability at semantic web. We automatically construct an OWL ontology from RDB schema using direct mapping rules. The mapping rules provide the basic rules for generating RDF triples from RDB data even for column contents null value, and enable semantic query engines to answer more relevant queries. Then we rewriting SPARQL query from SQL by translating SQL relational algebra into an equivalent SPARQL. The proposed method is demonstrated with examples and the effectiveness of the proposed approach is evaluated by experimental results.**

Keywords-Relational database; Ontology; SQL algebra; SPARQL; Mapping rule

I. INTRODUCTION

The semantic web is one of the most important research fields that came into light recently. It is a vision of the W3C [1] to make web information readable not only by human beings but also by machines. Ontology is a key enabling technology for the semantic web applications. It plays a crucial role in solving the problem of semantic heterogeneity of heterogeneous data sources [2] and contributes to improve system interoperation. The W3C has recommended several formats of languages for representing web ontology, such as resource description framework (RDF) [3], RDF Schema [4], and web ontology language (OWL) [5]. Moreover, the semantic query languages for web ontology is recommended, such as SPARQL [6, 7]. Since SPARQL is the standard query language for the RDF data model, which is supported by the Jena API [8], we accordingly use SPARQL query in this study.

Currently, the bulk of web content "deep web" is stored in relational databases (RDBs) with no near future vision for massive global RDB to RDF triple store migration. The success of semantic web depends on its ability to access RDBs and their content by semantic methods. Therefore, it is highly desirable to generate ontology from RDB resources mainly in order to publish data as RDF/OWL on the web and to combine a relational data with existing RDF/OWL for data integration. Recent approaches have been developed by W3C RDB2RDF Working group and proposed a basic transformation of RDB data to ontology (RDF) [9, 10]. The RDF can be queried through semantic query SPARQL [6, 7, 11] to provide a semantic query on RDF data.

Though integrating a database with the semantic web is a hard task to conduct, several important problems remain to be investigated. Some of the primary obstacles in integrating RDBs with semantic web are that, how an ontology can be constructed automatically from RDBs as RDF/OWL. Being an important step towards realizing benefits of semantic web research, and how the user can be assisted to formulate queries in order to retrieve more accurate information. A lot of difficulties exist in generating ontology from RDB or querying, including unclear generation approaches, query formulation, manage and query data stored in RDF files, determination of how to retrieve the transformation data, analysis of RDB and ontology, and their similarities, and dealing with relationships and null values.

In this paper, we identify and discuss the problem of direct mapping RDB to ontology including querying relational data and its ontology using semantic query (SPARQL). Two basic challenges make the problem interesting. Firstly, it is imperative for the community to develop fully automated method for bridging relational database content and the semantic web using ontology in the form of OWL/RDF data. Secondly, it is extremely difficult to express queries against graph structured ontology in the relational query language SQL.

The goal of this paper is to propose a novel approach for automatically building ontology (RDF graph with OWL vocabulary) from RDBs (Schema and data) and manage querying semantically on generated ontology. In order to accomplish an alternative for common query (SQL) on RDB data, the combination of ontology (OWL/RDF graphs) and an exemplary semantic query language (SPARQL) are investigated. Our main contributions in this paper can be summarized as follows.

- We propose a direct mapping rules to construct an ontology schema and data from RDB (even for column contents null value).
- We propose a query transformation approach by translating SQL relational algebra into an equivalent semantic query (SPARQL).
- We test the proposed approach on an RDB that contains an important concept of RDB scenarios, and demonstrate the effectiveness of the approach with

examples and experimental analysis. Every relational algebra query on an RDB data can be translated into an equivalent SPARQL query on ontology instance, which indicates that there are no any lose of information during the transformation process.

The rest of the paper is organized as follows. Section II presents related work. Section III and IV describe our approach, which construct ontology, generating RDF triples, and enable query on RDF triples. Experimental analysis is provided in Section V. Finally, Section VI concludes this paper with the future work.

II. RELATED WORK

Several efforts are made to integrate RDBs with the semantic web. During the last decade, data hosted in RDBs accessible to the semantic web has been an active field of research. Usually, an ontology model is constructed from RDB model, and the contents of the RDB are transformed to generate ontology instances [12, 13]. Li, et al. [14] proposed an automatic ontology learning approach to acquire OWL ontology from RDB automatically by using a group of rules, extracted ontology in an RDB using ER Data Model. This procedure has a disadvantage of losing the information because only the schema structure of an RDB has been used therefore, actual data is not utilized. On the other hand Astrova, et al. [15] proposed a novel approach for automatic transformation of RDBs to ontologies, where the quality of transformation is also considered. An RDB is written in SQL, and an ontology is written in OWL. The approach suffers from many shortcomings such as neglecting the formal definition which lead to ambiguous transformation rules. While, Zhang and LI [16] presented a method for automatic ontology building using the RDB resources to improve the efficiency. This method firstly, maps the analysis of ontology and database, secondly constructs rules of ontology elements from RDB. Then the practical experiments prove the method and system feasibility, however, this method ignores some tables that express association data, which could not be counted in the concepts. It should be mentioned that the previous studies did not deal with null-valued data through data conversion of RDB to RDF, and they did not investigate the transformation of query on RDF triples.

Another work dealing with Triplify Auer, et al. [17] offers a Linked Data publishing interface and provides a simplistic approach to publish RDF from RDB. Recently, the W3C RDB2RDF Working Group proposed a standard mapping language, called R2RML [18], to express RDB-to-RDF mappings. These approaches may require expert for complete mapping of RDB to the existing ontology, particularly to avoid problems that occur during mapping constraints.

The database queries, which are based on concepts, properties and instances defined in an ontology and that return semantically relevant results are referred to semantic query. There are several possibilities for querying ontologies from RDBs. RDF query language SPARQL [6, 7] provides a semantic query on RDF data, which focuses in transforming traditional SQL queries into RDF query languages. D2R Server [19] is engine that directly maps the RDB into RDF and uses D2RQ mappings to translate requests from external applications to SQL queries on the RDB. Lee and Sohn [20] presented a framework which can automatically construct an ontology from an RDB schema and can be clearly identified the semantic relations between data through the ontology construction. The constructed ontology help to understand data structure and acts as an assistant tool to efficiently query the data from RDB. Ranganathan and Liu [21] were defined three types (direct, inferred, and related results) of semantically relevant results based on how results are obtained and their relationship to the semantic query. The end-user issues semantic queries based on ontology concepts and these queries are mapped onto plain syntactic SQL queries.

The problem of query rewriting considered how to reformulate a query expressed in SPARQL over mediated schema into an equivalent SQL query targeting the underlying RDB. Cyganiak [22] discussed the transformation of SPARQL to relational algebra and outlines a set of rules to establish the equivalence between this algebra and SQL. Their study describes operators such as selection and inner join implemented over RDF and correlates RDF relational algebra to SQL. Their approach lacks the nested OPTIONAL pattern problem. Recently, the ontop system Rodriguez-Muro, et al. [23] also enables SPARQL queries to RDF views of RDBs by translating SPARQL to datalog programs, which are rewritten and translated to SQL.

Compared with existing approaches, our work is quite different in terms of an integrated method. For example, we extract ontology schema from RDB schema, transfer the contents of RDB (considering null-values) to RDF triples, and enable applications to query on RDF triples by creating new rules using SPARQL query corresponding to SQL query on RDB instances. The strength of our work it includes the important concepts of RDB, such as constraints, relationships, and null-values for all phases of the approach that are demonstrated with examples.

III. RULES FOR GENERATING ONTOLOGY FROM RDB

The contents of this part are represented by running examples, which includes the important cases, such as relationships of RDB as shown in Fig. 1.

A. *Rules for constructing ontology from RDB schema*

This step maps RDB schema to an ontology, which provides the basic rules for generating RDF triples from RDB data. Firstly, we define some predicates that will be used in this work as follows. Identify relationship between two tables:

$$\begin{aligned} &TB_i(A_1,...,A_n),TB_k(B_1,...,B_n),TB_i.A \rightarrow^{fk} TB_k.B \\ &\qquad \rightarrow \mathbf{HasRelationship}(TB_i,A,TB_k,B) \\ &\qquad \Rightarrow \mathbf{IsRelationship}(TB_i.A,TB_k.B) \qquad (1) \end{aligned}$$

i,k *are name of tables* TB *and* $A \in attr(TB_i), B \in attr(TB_k), TB_i, TB_k \in TB \subseteq RDBS.$

Identify binary relation:

$$\begin{aligned} &\mathbf{IsRelationship}(TB_b.A_1,TB_i.B),\mathbf{IsRelationship}(TB_b.A_2,TB_k.C), \\ &pk_2(A_1,A_2,TB_b),\ pk_1(B,TB_i),\ pk_1(C,TB_k) \\ &\qquad \rightarrow \mathbf{BinaryRel}(TB_b,A_1,A_2,TB_i,B,TB_k,C) \\ &\qquad \Rightarrow \mathbf{ISBinaryRel}(TB_b) \qquad (2) \end{aligned}$$

$: (b,i,k$ *are name of tables*$, b \neq i, b \neq k$ *and* $i \neq k)$*and* $TB_b, TB_i, TB_k \in TB \subseteq RDBS.$

Then the mapping process is done progressively based on the rules in Table I.

TABLE I. RULES OF CONSTRUCTING ONTOLOGY FROM RDB SCHEMA

Rule	DB Concept	Case Condition	OWL/RDF
1	Table (T)	!IsBinRel(T)	Owl:Class(T)
2	Column T(A)	**!**(PK(A,T)∨FK(A,T))	Owl:DatatypePropert ,rdfs:domain(T), rdfs:range(xsd)
3	PK constraint	PK(A,T)	Owl:InverseFunction Property(A1) RestrictionProp(A,minCard,xsd^^int 1)
4	1:1 (Fig. 1a)	T1(A1..An)∞T2(B1.. Bn),FK(A1,T1),PK(B1,T2),A1≠null,!(IsBinRel(T1)∨ IsBinRel(T2))	OjectProperty(A1,T1 $_{domain}$,T2$_{raing}$), RestrictionProp(A1,hasValue,T2), RestrictionProp(A1, minCard,xsd^^int 1)
5	1:m (Fig. 1b)	T1(A1..An)∞T2(B1.. Bn),FK(A1,T1),PK(B1,T2),valueOf(T1.A1,From,T2.B1),A1≠null,!(IsBinRel(T1) ∨IsBinRel(T2))	ObjectProperty(A1,T1$_{domain}$,T2$_{raing}$) ,RestrictionProp(A1,allValueFrom,T2), RestrictionProp(A1, minCard,xsd^^int 1)
6	m:n (Fig. 1c)	T1(A1,A2),T2(B1..Bn),T3(C1..Cn),IsBinRel(T1)→BinRel(T1, A1,A2,T2,B1,T3,C1)	OBP(B1,T2$_{domain}$,T3$_{raing}$),RestrictionProp(B1,allValueFrom,T3), RestrictionProp(B1, minCard,xsd^^int 1), OBP(C1,T3$_{domain}$,T2$_{raing}$),RestrictionProp(C1,allValueFrom,T2), RestrictionProp(C1, minCard,xsd^^int 1)

B. *Generating RDF triples*

The RDF triples are generated from RDB instance, in order to establish simple way and to access RDF triples using semantic search technologies. If a table T is mapped to the class then all rows of the table are transformed to the instance of RDF graphs. Each column in table T can be transformed to the data properties of the instance unless its value is null. The properties generated from foreign key columns are linked between classes. To ensure the uniqueness of resources, we form the IRI of the triples by combination of the name space NS (base IRI), table name, and primary key values.

IV. REWRITING SPARQL FROM SQL ALGEBRA

The semantic web applications need to be accessing relational database contents by semantic methods. Currently, SPARQL is a W3C recommendation, and has become the standard language for querying RDF data. Assume a given relational instance I over table (s) TB and ontology instance Io over class (es) CLS . We proof and explain that, for every relational algebra query $Q(I(TB))$, there is a SPARQL query $Q^o(Io(CLS))$ such that for every instance I of TB (possibly including null values) satisfying the following function:

$$Q(_{I(TB)}) \to Q^\circ(_{Io(CLS)}) : I(TB) = \{rw_1, ..., rw_n\},$$
$$rw_i = \{rw_i.A_1, ..., rw_i.A_{nc}\},$$
$$Io(CLS) = \{gr_1, ..., gr_n\}, gr_i = \{t_{id}, t_1.A_1, ..., t_{nc}.A_{nc}\} \qquad (3)$$

Where I is a set of rows $rw_1,...,rw_m$ over TB that denoted by $I(TB)=\{rw_1,...,rw_m\}$ for all attributes in TB . The notation $rw_i.A_i$ refers to the value of a row rw_i in a column A_i . Io is a set of triples $t_1,...,t_m$ of the graph gr_r over CLS that denoted by $Io(CLS)=\{t_1,...,t_m\}$ for all datatype properties in CLS. The notation $t_1.A_1$ refers to the value of triple t_1.in a property A_1. The t_{id} is the first triple of row that its type is the class that determine the table from which the triple is generated. The operator OPTIONAL is used to avoid the loss of information, because our rules that generated RDF triples from RDB data does not translate null-values. Therefore, the rules of semantic query in this section handles null-value in query expressions through two operators BOUND and OPTIONAL. Then the equivalent SPARQL queries for the relational algebra operations: Selection (σ), Projection ($\prod$), Rename (ρ), Union ($\cup$), Difference ($\backslash$), Natural Join (∞), Left Join ($\propto$), and binary relation were defined.

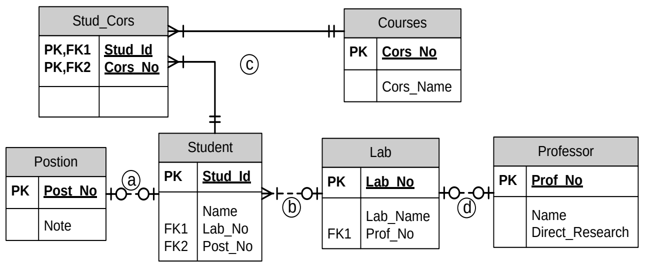


Figure 1. RDB laboratory (RDBLAB)

A. *Rules for fundamental operations of relational algebra*

***Rule1*(Selection(σ)):** The selection σ is a unary operation in relational algebra. The expression

$$\sigma_p(TB_1 \times .. \times TB_n) : n \geq 0, p \subseteq \{A_i\theta A_k, A_i\theta v, p_1\phi p_2,$$
$$IsNull(A_i), IsNotNull(A_i), \text{like*}, \text{IN}\}, \theta \in \{=, \neq, <, \leq, \geq, >\},$$
$$\phi \in \{\&\&, \|\}, A_i, A_k \in att(TB), v \textit{ is a constant value.}$$

Where the P stands for an expression condition in the set $\{A_i\theta A_k, A_i\theta v, p_1\phi p_2, IsNull(A_i), IsNotNull(A_i), Like, IN\}$, θ is a binary operation of the set $\{=,\neq,<,\leq,\geq,>\}$, ϕ is a logical operation $\{\text{and } \&\&, \text{ or } \|\}$, and p_1, p_2 are expiration condition. Therefore, we need to consider all the cases to define a query Q° to satisfy the defining condition (3).

1. $\sigma_{A_i=v}(TB) \to (GP$ **FILTER** $(?A_i = v))$ or (GP **FILTER** regex$(?A_i, v)) : A_i \in att(TB), ?A_i \in \text{var}(GP)$ **(R_1.1)**
2. $\sigma_{A_i=v1 \text{ and } A_k \geq v2 \textit{ or } A_d > 01-01-2015}(TB) \to (GP$ **FILTER** $(?A_i = v1 \;\&\&\; ?A_k \geq v2 \;\|\; ?A_d > \text{"2015-01-01"\^\^xsd:date}))$ **(R_1.2)**
3. $\sigma_{IsNotNull(A_i)}(TB) \to (GP$ **FILTER** $(\text{bound}(?A_i)))$ **R_1.**3)

GP is a graph pattern expression constructed from triple pattern $TP \in (IR \cup BN \cup L \cup V) \times (IR \cup V) \times (IR \cup L \cup V)$, where BN is blank node, L is literal, and V is variable.

SPARQL FILTER restricts the solutions of a query by imposing constraints on values of bound variables.

Rule2(Projection $\prod$ ***):*** The projection $\prod$ is a unary operation in relational algebra that selects of the relevant attributes of a relation. Let the expression

$Q = \prod_{TB.A_1,\ldots,TB.A_n}(TB)_{Order\ by\ TB.A_i\ [ASC|DESC]}$,
and $n \geq 0, i \in \{1,\ldots,n\}$. Then the equivalent query Q° to satisfy the defining condition (3) can be defined as:

$$Q = \prod_{TB.A_1,\ldots,TB.A_n}(TB)_{\text{Order by TB.Ai [ASC|DESC]}} \rightarrow Q^\circ =$$

Select $?TB.A_1 \ldots ?TB.A_n$ *where* $\{GP\}$ order by [ASC|DESC](?TB.Ai) **(R_2)**

***Rule3*(Rename ρ)**: A rename ρ is a unary operation in relational algebra that renames one column to another name and projects all columns of Q. Let the expression

$Q = \prod_{\rho(TB.A_1,\ldots,TB.A_n)\mapsto(B_1,\ldots,B_n)}(TB)$. Then the equivalent query Q° to satisfy the defining condition (3) can be defined as:

$$Q = \prod_{\rho(TB.A_1,\ldots,TB.A_n)\mapsto(B_1,\ldots,B_n)}(TB) \rightarrow Q^\circ =$$

Select $(?TB.A_1$ **AS** $?B_1) \ldots (?TB.A_n$ **AS** $?B_n)$ *Where*$\{GP\}$ **(R_3)**

The SPARQL expression equivalence in this rule is hold because the rename operator in relational algebra renames one column to another and projects all Q columns.

***Rule4*(Union $\cup$)**: A union $\cup$ is a binary operator that combines the result-set of two or more projects($\prod$)-Select statements. Let the expression

$$Q = \prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1) \cup \ldots \cup \prod_{TB_n.A_1,\ldots,TB_n.A_{nc}}(TB_n), n > 1$$

Then the equivalent query Q° to satisfy the defining condition (3) can be defined as:

$$Q = \prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1) \cup \ldots \cup \prod_{TB_n.A_1,\ldots,TB_n.A_{nc}}(TB_n)$$
$\rightarrow Q^\circ =$ *Select * where* $\{\{$*Select* $?TB_1.A_1 \ldots ?TB_1.A_{nc}$ *where*
$\{GP\}\}$ **UNION**...**UNION**
$\{$*Select* $?TB_n.A_1 \ldots ?TB_n.A_{nc}$ *where* $\{GP\}\}\}$ **(R_4)**

***Rule5*(Difference $\setminus$)**: A difference $\setminus$ is an operator that minuses the result-set of two relations TB_1 and TB_2 where two relations should have the same attributes. The result of $TB_1 \setminus TB_2$ is a relation that contains all rows in TB_1 but not in TB_2. Let the expression

$$Q = \prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1) \setminus \prod_{TB_2.A_1,\ldots,TB_2.A_{nc}}(TB_2)$$

Then the equivalent query Q° to satisfy the defining condition (3) can be defined as follows:

$$Q = \prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1) \setminus \prod_{TB_2.A_1,\ldots,TB_2.A_{nc}}(TB_2) \rightarrow$$
Select $?A_1 \ldots ?A_n$ Where {**?x** a NS:TB$_1$. **?x** NS:TB$_1$.A$_1$?A$_1$....
FILTER NOT EXISTS{*?y* a NS:TB$_2$. *?y* NS:TB$_2$.A$_1$ **?x**.}} **(R_5)**

For example Q1:

$\mathbf{Q1} = \prod_{Lab.Lab\_No}(Lab) \setminus \prod_{Student.Lab\_No}(Student) \rightarrow$
SELECT ?Lab_No Where { ?x a NS:Lab. ?x NS:Lab.Lab_No ?Lab_No.
FILTER NOT EXISTS{?y NS:Student.Lab_No ?x. ?y a NS:Student.}}
order by ?Lab_No

B. Rules6 for a relational algebra join

The SQL join clause is used to combine rows from two tables or more, based on a common field between them.

***Rule6.1*(Natural join ∞)**: A natural join ∞ is a binary operator that combines rows from two tables or more $TB_1 \infty TB_2 \infty \ldots TB_n$, based on a common field between them. The result of $TB_1 \infty TB_2$ is a set of all combination rows in TB_1 and TB_2 that are equal on their common column names. Let the expression $Q = \prod_{TB_1.A_1,\mathbf{TB_1.A_i},\ldots,TB_1.A_{n1},TB_2.B_1,\mathbf{TB_2.A_i},\ldots,TB_2.B_{n2}}(TB_1 \infty TB_2)$, where $TB_1.A_i$ and $TB_2.A_i$ are common attributes. Then the equivalent query Q° to satisfy the defining condition (3) can be defined as:

$Q = \prod_{TB_1.A_1,\mathbf{TB_1.A_i},\ldots,TB_1.A_{n1},TB_2.B_1,\mathbf{TB_2.A_i},\ldots,TB_2.B_{n2}}(TB_1 \infty TB_2) \rightarrow$
Select $?TB_1.A_1 \ldots ?TB_1.A_{n1}\ ?TB_2.B_1 \ldots ?TB_2.B_{n2}$
Where $\{?x\ a\ NS:TB_1.$ **Optional**$\{?x\ NS:TB_1.A_1\ ?TB_1.A_1.\}\ldots$
Optional$\{?x\ NS:TB_1.A_{n1}\ ?TB_1.A_{n1}.\}$
$\mathbf{?x\ NS:TB_1.A_i\ ?TB_1.A_i.\ ?TB_1.A_i}\ a\ NS:TB_2.$
Optional$\{\mathbf{?TB_1.A_i}\ NS:TB_2.B_1\ ?TB_2.B_1.\}\ldots$
Optional$\{\mathbf{?TB_1.A_i}\ NS:TB_2.B_{n2}\ ?TB_2.B_{n2}.\}\}$ **(R_6.1)**

The graph pattern GP {?x NS:TB$_1$.A$_i$?TB$_1$.A$_i$. ?TB$_1$.A$_i$ a NS:TB2.} contains object property (NS:TB$_1$.A$_i$) represented by the variable ?TB$_1$.A$_i$, which used to connect between two classes NS:TB$_1$ and NS:TB$_2$. The **OPTINAL** operator for all properties of class was used (to avoid lose triples of property values) except the common property (to ensure that its value is **not-null)**. If we change the rule condition using **BOUND** operator in part $(?x\ NS:TB_1.A_i\ ?TB_1.A_i)$ to $\{?x$ $a\ NS:TB_1\{\ldots$***Optional***$\{?x\ NS:TB_1.A_i\ ?TB_1.A_i.\}$ ***FILTER*** $($***Bound***$(?TB_1.A_i))\}\ldots\}$, the same result was observed. The following SPARQL query example (Q2):

$\mathbf{Q2} = \prod_{Student.Stude\_Id,Student.Name,Lab.Lab\_Name}(Student \infty Lab) \rightarrow$
Select ?Stud_Id ?name ?lab_no ?lab_name **Where**
{?x a NS:Student. **Optional**{?x NS:Student.Stud_Id ?Stud_Id.}
Optional{?x NS:Student.Name ?name.}
?x NS : Student.Lab_No ?lab_no. ?lab_no a NS:Lab.
Optional{ **?lab_no** NS:Lab.Lab_Name ?lab_name.} } order by ?Stud_Id

***Rule6.2*(Left Join $\propto$)**: Let the expression

$$Q = \prod_{TB_1.A_1,\mathbf{TB_1.A_i},\ldots,TB_1.A_{n1},TB_2.B_1,\mathbf{TB_2.A_i},\ldots,TB_2.B_{n2}}(TB_1 \propto TB_2),$$

$TB_1.A_i$ and $TB_2.A_i$ are common attributes**.** Then the equivalent query Q° to satisfy the defining condition (3) can be defined as follows:

$Q = \prod_{TB_1.A_1,\mathbf{TB_1.A_i},\ldots,TB_1.A_{n1},TB_2.B_1,\mathbf{TB_2.A_i},\ldots,TB_2.B_{n2}}(TB_1 \propto TB_2) \rightarrow$
Select $?TB_1.A_1 \ldots ?TB_1.A_{n1}\ ?TB_2.B_1 \ldots ?TB_2.B_{n2}$
Where $\{?x\ a\ NS:TB_1.$ **Optional**$\{?x\ NS:TB_1.A_1\ ?TB_1.A_1.\}\ldots$
Optional$\{?x\ NS:TB_1.A_{n1}\ ?TB_1.A_{n1}.\}$

$$\textbf{Optional}\left\{\begin{array}{l}\mathbf{?x\ NS:TB_1.A_i\ ?TB_1.A_i.\ ?TB_1.A_i}\ a\ NS:TB_2.\\ \qquad \textbf{Optional}\{\mathbf{?TB_1.A_i}\ NS:TB_2.B_1\ ?TB_2.B_1.\}\ldots\\ \textbf{Optional}\{\mathbf{?TB_1.A_i}\ NS:TB_2.B_{n2}\ ?TB_2.B_{n2}.\}\end{array}\right\}\}$$

(R_6.2)

For example Q3:

$\mathbf{Q3} = \prod_{Student.Stude\_Id,Student.Name,Lab.Lab\_Name}(Student \propto Lab) \rightarrow$
Select ?Stud_Id ?name ?lab_no ?lab_name
Where {?x a NS:Student.
Optional{?x NS:Student.Stud_Id ?Stud_Id.}
Optional{?x NS:Student.Name ?name.}

$$\textbf{Optional}\left\{\begin{array}{l}\textbf{?x NS : Student.Lab\_No ?lab\_no.}\\ \ \textbf{?lab\_no}\ \text{a NS:Lab.}\\ \textbf{Optional}\{\text{ ?lab\_no NS:Lab.Lab\_Name ?lab\_name.}\}\end{array}\right\}\}$$

C. *Rules7 for a binary relation*

Assume that Q is a query algebra over binary relation. Then the equivalent query Q° to satisfy the defining condition (3) can be defined as follows:

$$Q^{\circ} = Select\ ?A\ \ ?B$$
$$Where\ \{\mathbf{?A}\ \ a\ \ NS:TB_1.\ \ \mathbf{?A}\ \ NS:TB_1.B\ \ \mathbf{?B}.$$
$$\mathbf{?B}\ \ a\ \ NS:TB_2.\ \ \mathbf{?B}\ \ NS:TB_2.A\ \ \mathbf{?A}.\}\quad \mathbf{(R\_7)}$$

The variables (?A and ?B) of this rule show the important positions for conditional clauses to obtain the equivalent SPARQL query according to expression of the binary relation algebra. As well to get the same results returned by expression of the relational algebra on the RDB. An example of this rule is Q4 that represents the binary relation in SPARQL query.

```
Q4 = Select ?Stud_Id ?name ?Cors_No  ?Cors_Name
    Where{ ?A  a  NS : Student. ?A  NS : Student.Cors_No ?B.
              ?A NS:Student.Stud_Id ?Stud_Id.
              ?A NS:Student.Name ?name.
              ?B  a  NS : Courses.   ?B  NS : Courses.Stud_Id ?A.
?B NS:Courses.Cors_No ?Cors_No.
?B NS:Courses.Cors_Name ?Cors_Name.}
```

From the previous analysis, it can be clearly observed that our rule satisfies the condition definition (3). Moreover, the transformation rules in Section III and IV designed in clear forms and keep the tracks of attribute keys in the tables. Therefore, these rules can be extended to generate RDB from ontology.

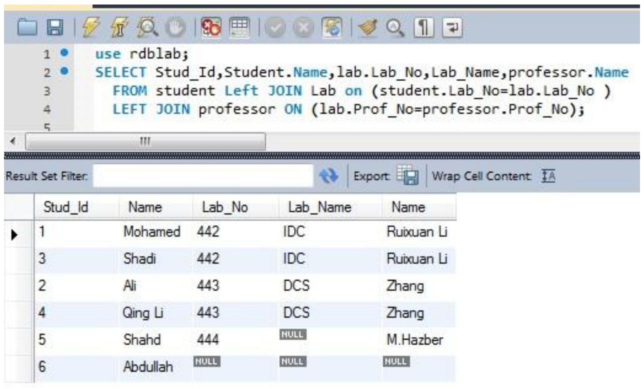
V. EXPERIMENTAL ANALYSIS

The information retrieval system was implemented in the platform of Windows 7 (32-bit) operating system with the specification of CPU Intel® Core™ i5-2410M 2.30GHz, RAM 6GB. In order to validate the efficiency of our rules in terms of quantitative, the dataset of RDB (RDBLAB) has been increased to include 70000 rows. In the generated ontology, RDF triples have 442668 triples. Table II shows the number of RDBLAB row tables, null-values in each column tables and number of rows in which null-values appear during the relationship between the tables. In this table, the null-values reflect the size of data that are not lost when our rules are used. Moreover, the table shows the numbers of tuple classes are corresponding to the row tables.

To reflect the validity of our approach, an extra example (Q5 and Q6) were added.

$$Q5: \prod_{Stud\_Id,Lab.Lab\_No,Prof.Name} ((Student \propto Lab) \propto Professor)$$

This query represents LEFT JOIN condition between three tables Student, Lab, and Professor in SQL query, to obtain the stud_Id and name, lab name, professor name of the all students who have /or have not the place in the lab and professors of the lab. Fig. 2 shows the corresponding SQL query relational algebra query and the results returned by the execution through RDBMS (MySQL).



```
use rdblab;
SELECT Stud_Id,Student.Name,lab.Lab_No,Lab_Name,professor.Name
  FROM student Left JOIN Lab on (student.Lab_No=lab.Lab_No )
  LEFT JOIN professor ON (lab.Prof_No=professor.Prof_No);
```

Stud_Id	Name	Lab_No	Lab_Name	Name
1	Mohamed	442	IDC	Ruixuan Li
3	Shadi	442	IDC	Ruixuan Li
2	Ali	443	DCS	Zhang
4	Qing Li	443	DCS	Zhang
5	Shahd	444	NULL	M.Hazber
6	Abdullah	NULL	NULL	NULL

Figure 2. LEFT JOIN query $((Student \propto Lab) \propto Professor)$ and the result returned by DBMS.

Therefore, the SPARQL corresponding to above query is as follows:

```
SELECT   ?Stud_Id ?name ?lab_no ?lab_name ?prof_name
WHERE   { ?x  a  NS:Student.
   optional{?x NS:Student.Stud_Id ?Stud_Id.}
   optional{?x NS:Student.Name ?name.}
   optional{?x  NS:Student.Lab_No ?lab_no. ?lab_no a  NS:Lab.
              optional{ ?lab_no NS:Lab.Lab_Name ?lab_name.}
  optional{ ?lab_no NS:Lab.Prof_No  ?prof_no. ?prof_no a NS:Professor.
        optional{ ?prof_no NS:Professor.Name ?prof_name.}}} order by ?Stud_Id
```

The returned results are shown in Fig. 3. Whereas SQL query Q6 can be represented by the following formula.

$$Q6 = \prod_{\rho(\text{Stud\_Id,Name,'Stud'})\mapsto(\text{id,name,type})} \left(Student\right) \cup$$
$$\prod_{\rho(\text{Lab\_No,Lab\_Name,'Lab',})\mapsto(\text{id,name,type})} \left(Lab\right) \cup$$
$$\prod_{\rho(\text{Prof\_No,Name,'Prof'})\mapsto(\text{id,name,type})} \left(Professor\right)$$

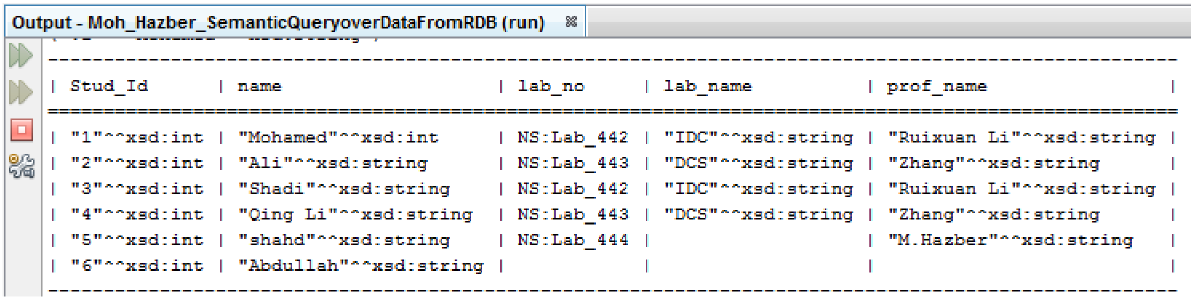
Therefore, the SPARQL corresponding to above query is as follows:

```
SELECT  * WHERE {
     { SELECT (?Stud_Id As ?id)  (?Name As ?name)  ('Stud' AS ?type)
WHERE  { ?S  a  NS:Student.
            optional{?S NS:Student.Stud_Id ?Stud_Id.}
            optional{?S NS:Student.Name ?Name.}
        }} union
    {SELECT (?Lab_No AS ?id) (?Lab_Name AS ?name) ('Lab' AS ?type)
WHERE {?lab_no a NS:Lab.
        optional{?lab_no NS:Lab.Lab_No ?Lab_No.}
       optional {?lab_no NS:Lab.Lab_Name ?Lab_Name.}
   }} union
   {SELECT  (?Prof_No AS ?id)  (?Name AS ?name) ('Prof' AS ?type)
WHERE{?prof_no a  NS:Professor.
        optional{?prof_no NS:Professor.Prof_No ?Prof_No.}
       optional{?prof_no NS:Professor.Name ?Name.}
  }}  } order by ?id
```

Interestingly, the same results were obtained after execute the Q6 and its SPARQL equivalent (Table IV).

TABLE II. RDBLAB TABLE ROWS WITH NULL-VALUE AND ONTOLOGY CLASS TUPLES.

Tables	Rows	Columns of null-value	Rows of null-value	SPARQL TP used to return class tuples	Returned tuples
Student	51000	Lab_No=7614 Age=2500	Students with lab_no is null=7614 Students with age is null=2500 Students with lab_no is not null and lab.lab_name is null=6448 Students with lab_no is not null and lab.prof_no is null=3378	{?x a NS:Student}	51000
Lab	1000	Lab_Name=199 Prof_No=111	Rows of lab used by students =33 Rows of lab used by students with lab_name is null=6	{?x a NS:Lab}	1000
Courses	100			{?x a NS:Courses}	100
Stud_Cors	17800			Q4	17800
Professor	100			{?x a NS:Professor}	100


Output - Moh_Hazber_SemanticQueryoverDataFromRDB (run)

```
---------------------------------------------------------------------------------------------------
| Stud_Id       | name                      | lab_no      | lab_name            | prof_name             |
===================================================================================================
| "1"^^xsd:int  | "Mohamed"^^xsd:int        | NS:Lab_442  | "IDC"^^xsd:string   | "Ruixuan Li"^^xsd:string |
| "2"^^xsd:int  | "Ali"^^xsd:string         | NS:Lab_443  | "DCS"^^xsd:string   | "Zhang"^^xsd:string      |
| "3"^^xsd:int  | "Shadi"^^xsd:string       | NS:Lab_442  | "IDC"^^xsd:string   | "Ruixuan Li"^^xsd:string |
| "4"^^xsd:int  | "Qing Li"^^xsd:string     | NS:Lab_443  | "DCS"^^xsd:string   | "Zhang"^^xsd:string      |
| "5"^^xsd:int  | "shahd"^^xsd:string       | NS:Lab_444  |                     | "M.Hazber"^^xsd:string   |
| "6"^^xsd:int  | "Abdullah"^^xsd:string    |             |                     |                          |
---------------------------------------------------------------------------------------------------
```

Figure 3. Query result on Netbeans console $((Student \propto Lab) \propto Professor)$.

This query (Q6) represents projection, rename, and union to combine three tables Student, Lab, and Professor in SQL query in the same list. By applying our approach there is no data loses, even for the null values. Moreover, the combinations of ontology and SPARQL query have the ability to provide the same results of RDB using SQL query algebra.

To emphasize the accuracy of our rules that used in Section IV, we apply all the previous queries (Q1-Q6) on the new dataset. To show the significance of our approach, additional SPARQL (Q`) queries are modified from our original queries SPARQL (Q°) and represented in Table III. These SPARQL (Q`) queries are then used for comparative and to reflect the volume of data loss. The queries (Q1-Q6) of SQL(Q) are applied on RDBLAB data using RDBMS (MYSQL 5.6), while the queries of SPARQL(Q°) and SPARQL(Q`) are applied on RDF triples generated from RDBLAB using our system. The results of queries are shown in Table IV.

The quantitative analyses of dataset shown in Table IV are represented in Figs. 4 and 5. By using our rules SPARQL(Q°) the same results are obtained from SQL(Q) and there are no data losses from SQL(Q) compared to SPARQL(Q`) (Fig. 5). All these results together reflect the high accuracy and significance of our rules.

TABLE III. SPARQL(Q`) QUERIES.

Query	Conditions used to modify Q` from original examples of Q°
Q1	If {?x NS:Lab.Lab_Name ?lab_name } added to get lab name without OPT
Q2	If the OPT deleted from the triple pattern {?lab_no NS:Lab.Lab_Name ?lab_name.}
Q3	If the OPT that used for LEFT OUTER JOIN is deleted
Q4	If {?Stud_Id NS:Student.Age ?age.} added to get the age of students without OPT
Q5	If the two OPTs that used for LEFT OUTER JOIN are deleted.
Q6	If all OPTs are deleted from triple patterns

TABLE IV. RESULTS OF QUERIES.

	Q rows	Q° tuples	Q` tuples
Q1	967	967	774
Q2	43386	43386	36938
Q3	51000	51000	43386
Q4	17800	17800	15300
Q5	51000	51000	40008
Q6	52100	52100	51901

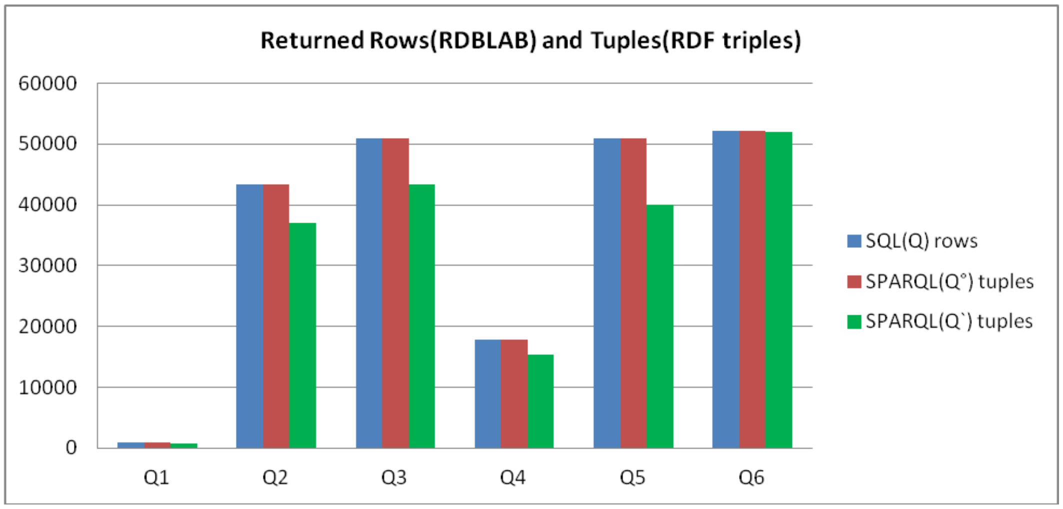


Figure 4. Comparing between SQL(Q), our approach SPARQL(Q°) and SPARQL(Q`) results.

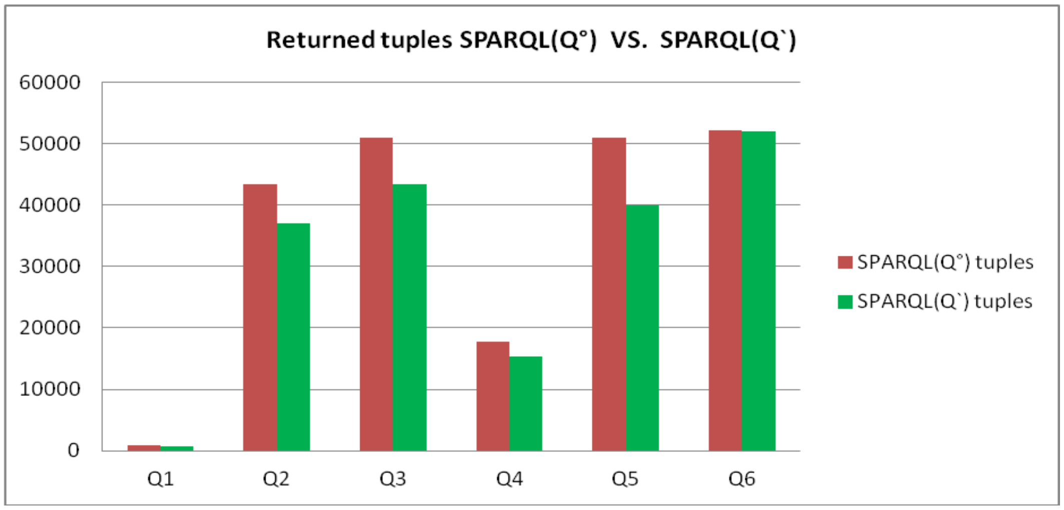


Figure 5. Comparing between our approach SPARQL(Q°) and SPARQL(Q`) results.

VI. CONCLUSIONS

Study on ontology construction from RDB is becoming increasingly widespread in the computer science community, which includes a definition of domain metadata, relationships and knowledge of the ontology schema to assist in the query formulation process. In this paper, we proposed a new approach for direct mapping of RDB (schema, data, and SQL query) to semantic web ontology (OWL, RDF, and SPARQL). The semantic query in a RDB is simulated and implemented using SPARQL. SPARQL can be considered as a real alternative to the commonly used SQL access to relational databases. The effectiveness of the proposed approach is demonstrated with examples and experimental analysis.

Our approach does not explain in details on how to rewrite the equivalent SPARQLs from the nested SQL queries which will be considered in the future works.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under grants 61173170, 61300222, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2013QN120, and Science and Technology Support Program of Hubei Province under grant 2014BCH270.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, pp. 28-37, 2001.

[2] O. Vasilecas, D. Bugaite, and J. Trinkunas, "On Approach for Enterprise Ontology Transformation into Conceptual Model," *in Proc. International Conference on Computer Systems and*

Technologies, University of Veliko Tarnovo, Bulgaria, 2006, pp. IIIA.23-1- IIIA.23-6.

[3] R. Cyganiak, D. Wood, and M. Lanthaler. (2014). *RDF 1.1 concepts and abstract syntax*. Available: http://www.w3.org/TR/rdf11-concepts/

[4] D. Brickley, R. Guha, and B. McBride. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. Available: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[5] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. (2009). *OWL 2 Web Ontology Language: Profiles*. Available: http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/

[6] E. Prud'hommeaux and A. Seaborne. (2008). *SPARQL Query Language for RDF*. Available: http://www.w3.org/TR/rdf-sparql-query/.

[7] S. Harris and A. Seaborne. (2013). *SPARQL 1.1 Query Language*. Available: http://www.w3.org/TR/sparql11-query/

[8] A. jena. (2015). *A free and open source Java framework for building Semantic Web and Linked Data applications*. Available: http://jena.apache.org/index.html

[9] A. Bertails and E. G. Prud'hommeaux, "Interpreting relational databases in the RDF domain," *in Proceedings of the sixth international conference on Knowledge capture*, Banff, AB, Canada, 2011, pp. 129-136.

[10] M. Arenas, A. Bertails, E. Prud, and J. Sequeda. (2012). *A Direct Mapping of Relational Data to RDF*. Available: http://www.w3.org/TR/rdb-direct-mapping/

[11] J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to RDF and OWL," *in Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 649-658.

[12] H. Mohamed, Y. Jincai, and J. Qian, "Towards Integration Rules of Mapping from Relational Databases to Semantic Web Ontology," *in Web Information Systems and Mining (WISM), 2010 International Conference on*, Sanya, China, 2010, pp. 335-339.

[13] S. H. Tirmizi, J. Sequeda, and D. Miranker, "Translating sql applications to the semantic web," *in Proceedings of the 19th international conference on Database and Expert Systems Applications(DEXA '08)*, Turin, Italy, 2008, pp. 450-464.

[14] M. Li, X.-Y. Du, and S. Wang, "Learning ontology from relational database," *in Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, Guangzhou, China, 2005, pp. 3410-3415.

[15] I. Astrova, N. Korda, and A. Kalja, "Rule-based transformation of SQL relational databases to OWL ontologies," *in Proceedings of the 2nd International Conference on Metadata & Semantics Research*, Ionian University, Corfu,Greece, 2007.

[16] L. Zhang and J. LI, "Automatic Generation of Ontology Based on Database," *Journal of Computational Information Systems*, vol. 7, pp. 1148-1154, 2011.

[17] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller, "Triplify light-weight linked data publication from relational databases," *in: Proc. of the 18th International Conference on World Wide Web*, Madrid, Spain, 2009, pp. 621–630.

[18] O. Souripriya Das, O. Seema Sundara, and R. Cyganiak. (2012). *R2RML: RDB to RDF mapping language*. Available: http://www.w3.org/TR/r2rml/

[19] C. Bizer and R. Cyganiak, "D2r server-publishing relational databases on the semantic web," *in Poster at the 5th International Semantic Web Conference (ISWC2006)*, Athens, GA, USA, 2006.

[20] H. J. Lee and M. Sohn, "DB schema based ontology construction for efficient RDB query," *in ACIIDS'12 Proceedings of the 4th Asian conference on Intelligent Information and Database Systems* - Volume Part II, Kaohsiung, Taiwan, 2012, pp. 341-350.

[21] A. Ranganathan and Z. Liu, "Information retrieval from relational databases using semantic queries," *in Proceedings of the 15th ACM international conference on Information and knowledge management*, Arlington, VA, USA, 2006, pp. 820-821.

[22] R. Cyganiak, "A relational algebra for SPARQL," *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, p. 35, 2005.

[23] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese, "Evaluating SPARQL-to-SQL translation in ontop," *in 2nd OWL Reasoner Evaluation Workshop (ORE 2013)-Collocated with DL 2013 Workshop*, Ulm, Germany, 2013, pp. 94-100.