6 authors, including:

Mohamed Hazber
Wuhan University
11 PUBLICATIONS **48** CITATIONS
SEE PROFILE

Bing Li
Wuhan University
121 PUBLICATIONS **1,119** CITATIONS
SEE PROFILE

Guandong Xu
University of Technology Sydney
245 PUBLICATIONS **1,821** CITATIONS
SEE PROFILE

Mohammed Mosleh
Dr. G. R. Damodaran College of Science, Coimbatore
4 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Short text View project

An Approach for Generation of SPARQL Query from SQL Algebra based Transformation Rules of RDB to Ontology

Mohamed A. G. Hazber [1*], Bing Li[1], Guandong Xu [2], Mohammed A. S. Mosleh[3], Xiwu Gu [4], and Yuhua Li[4]

[1]School of Computer Science, Wuhan University, Wuhan, China
[2]Advanced Analytics Institute, University of Technology Sydney, Australia
[3]School of IT & Science, Dr. G.R.Damodaran College of Science, Coimbatore, India
[4]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

*Correspondence: Email: moh_hazber@whu.edu.cn, moh_hazbar@yahoo.co.uk
Manuscript submitted August 31, 2018; accepted October 10, 2018.
doi: 10.17706/jsw.13.11.573-599

Abstract: Semantic web is a web of linked RDF data that can exchange and reuse data allowing for more use of traditional web documents. However, the huge amount of data on the web are still formed and stored in relational databases (RDBs), such data cannot be used directly via the Semantic Web. Consequently, construction of ontology (Semantic Web -side) from relational schema and data (RDBs - side) and querying of constructed ontology semantically are fundamental challenges for the development and integration of the Semantic Web from the data source (i.e. database). This paper proposes an approach for providing a formulated operation rules to express semantic queries against structured graph ontology in the relational query language SQL. This approach applied by rewriting SPARQL queries over generated ontology (i.e. RDF triples) corresponding to advantages of SQL relational algebra operation queries in RDBs and performed by two phases. The first phase focused on proposing and improving rules of extracting ontology directly from the important concepts in the relational database with considering database containing null-values to avoid data losses during the transformation process. The generated ontology represented in the form of OWL-RDFS/RDF triples to ensure its availability at Semantic Web, thus help semantic query engines to answer more queries. Furthermore, the first phase providing additional rules to generate the Internationalized Resource Identifiers (IRIs) for RDB schema and data. In the second phase, we proposed a set of rules inspired by fundamental operations of relational algebra (SQL algebra) for rewriting a relational algebra for SPARQL over RDBs that represented in RDF triples. In other words, translating SQL relational algebra operation queries into equivalent graph semantic queries (SPARQL). The proposed approach is demonstrated with examples, validated, implemented and compared with existing approach methods. The effectiveness of the proposed approach is evaluated by experimental results.

Key words: Semantic web, ontology, semantic query, SPARQL, transformation rule, relational database, SQL, relational algebra.

1. Introduction

The Semantic Web has become one of the most significant research fields that came into light recently. It is an idea of the W3C [1] to make web information understandable not only by human beings but also by machines. Ontology is basically for enabling technology to the semantic web applications and plays a crucial role in solving the problem of semantic heterogeneity of heterogeneous data sources [2]. Therefore, most researches focus on

the development of various technologies on semantic web. The W3C has recommended a number of languages for representing web ontology, such as resource description framework (RDF) [3] as standard language to represent data model, RDF Schema [4] as a schema of data, and web ontology language (OWL) [5], a formal language for authoring ontologies. Moreover, the semantic query language (i.e. SPARQL [6], [7]) for web ontology is recommended by W3C. SPARQL is the standard query language used for querying RDF data model, we accordingly use SPARQL query in this study.

Currently, the bulk of web data (i.e. deep web) is stored in RDBs with no near future vision for huge global RDB to RDF triple store migration. It can be noticed that the capacity to publish RDBs in the semantic web is significant not only for development of the latter, but also for increasing demand for the ability to effectively exchange this data and allowing search engines to return more relevant deep web search results [8]. One of the challenges in real world applications is how to make accessing data and sharing the existing knowledge in databases more efficient. That is mean there are important challenges in using RDB as an RDF data to enable web applications of accessing the RDBs. One of the studies was proved that Internet available databases, compared to the static web, contained up to 500 times more data and roughly 70% of websites are backed by RDBs [9]. Therefore, the success of the Semantic Web in this process depends on its ability to access RDBs and their content by semantic methods.

The continuous growth in the volume of published data on the web makes a challenge for providing some automatic mechanism to search and integrate information over the web, which is not possible on existing web. The majority part of these published data have come from RDBs. Therefore, it is highly desirable to produce ontology from relational database resources for publishing data as RDF/OWL on the web and combining a relational data with existing RDF/OWL for data integration. During the last decade several studies have been conducted to integrate a database with the semantic web and making data hosted in RDBs accessible to the semantic web. They providing methods and tools that expose or convert data in RDB as ontological data described in RDF. Recently, there are some issues existing methods of transforming relational module to ontology (OWL/RDF(S)) [10] and basic transformation methods of RDB data to ontology (RDF) [11], [12] . The RDF data model can be queried through SPARQL [6], [7] to provide a semantic query on RDF triples. The different features of existing approaches based on comparing of RDB-to-RDF mapping language were listed in [13].

However, semantic integration of relational data sources into the Semantic Web is not a trivial task and several important problems remain to be investigated. Some of the primary obstacles in integrating semantic web with RDBs are that, how ontology can be automatically constructed from RDBs as RDF/OWL triples, being a significant step towards realizing benefits of semantic web research, and how to formulate queries in order to retrieve more accurate information using SPARQL query. A lot of problems exist in constructing ontology from RDB or re-writing semantic querying corresponding to SQL query, including unclear ontology generation approaches, non-uniform methods in description of data from RDB by ontology, semantic query formulation, dealing with relationships and null values, manage and query data stored in OWL/RDF files.

Therefore, constructing ontology from RDBs and generating queries through ontologies are fundamental problems for the development of the semantic web and integrated with information sources. This paper aims to propose an approach for automatic ontology construction from RDBs and to participate in formulating semantic queries (SPARQL) corresponding to SQL query algebra. Moreover, to provide unified ontology and improve the quality of ontology, we added new complementary concepts to analyze RDB and ontology with their relation. Our main contributions in this paper can be summarized as follows.

(i) We propose a direct mapping rules for constructing ontology schema from RDB schema, and use these rules as a basis to build rules for generating RDF data model from RDB data (contain null

values). Therefore, RDB data become an integral part of the semantic web formatted that enable semantic query engines to answer more relative queries.

(ii) We propose a query transformation approach to show that generating RDF triples using our approach enables the semantic web applications accessing relational data. This approach applied by formulating translating rules SQL relational algebra into an equivalent semantic query (SPARQL).

(iii) We examine the performance of the proposed method on an RDB (have an important concept of RDB scenarios), and demonstrate the effectiveness of our approach practically by using examples and experimental analysis. The results of the queries are presented and demonstrated to be promising.

The rest of the paper is organized as follows: Section 2 introduces related work. Section 3 describes and analyzes the preliminary concepts of semantic web ontology (SWO) and relational database. Section 4 and 5 propose our approach for ontology construction, RDF triples generation, and expressing SPARQL queries against graph structured ontology in the relational SQL query (i.e. rewriting SPARQL queries corresponding to advantages of SQL query). Implementation, experimental analysis, and comparison are provided in Section 6. Our conclusion and future work direction are presents in the Section 7.

2. Related Work

In this section we discuss the previous works considering the following sub-division.

2.1. Transforming RDB to Ontology

In this section, we provide an overview of the previous effort solutions [14], [15], which aim to extract data model of ontology from an RDB schema (model) and to convert the relational data to the ontology instances. Buccella et al. [16] proposed the semi-automatic method to integrate several sources of information based on the use of ontologies. Every data source has a source ontology constructed in two phases: Producing initial ontology (OWL) from SQL-DDL and construction of source ontology, which allowed the domain experts for adding restrictions, classes, and properties to the initial ontology (OWL). The drawback in transformation rules of datatype properties have not include a domain and range, the not-null restriction was converted to the number restriction, and their translation was not capable of expressing the primary keys. Moreover, in the case of SQL-DDL code does not explain the minimal cardinality, to solve this case, domain experts needed to add this cardinality after the ontology was built. Li et al. [17] proposed an approach for the automatic ontology learning approach to develop OWL ontology from RDB using a set of rules and extracted ontology from an RDB using entity relationship (ER) Data Model. This method has a disadvantage of losing the information because only the RDB schema structure has been considered while the actual data is not utilized. On the other hand Shen et al. [18] is a semi-automatic approach that presented groups of semantic mapping rules to extract a global ontology as OWL from an RDB. They are classified as concepts, properties, restrictions and instances and used these rules to mapping RDB to ontologies in OWL, whereby the mapping and transferring can be performed semi-automatically. The rules of concepts, properties and restrictions represent the correspondence at the schema (metadata) level, which avoid the migrating the large amount of data. Another study carried by Astrova et al. [19], it has been widely cited by many approaches that proposing heuristics for mapping RDBs to ontologies. They proposed a method to automatically transform RDBs to ontologies, where the quality of transformation is also studied. This method is based on descriptive informal rules, which lead to ambiguous transformation rules. While Zhang and LI [20] presented a tool to generate ontology based on RDB resources, namely the ontology automatic generation system based on a relational database (OGSRD). This method firstly, mapping analysis of database and ontology. Secondly, building rules of an OWL ontology based on RDB, which are used to produce ontology classes, properties and axioms. Thirdly, designing and implementing the OGSRD. However, their tool disregarded some tables that express the association of data, which could not be counted in the RDB concepts.

2.2. Mapping an RDB to Existing Ontology

Approaches in this area indicating that a legacy RDB and ontology are already exist [21], [22]. The general goal is to create mappings between them, and / or populate the ontology with the RDB contents. For example Xu et al. [23] presented a practical approach for creating generic mappings between RDB (schema) and ontology (OWL). Ontological annotation is useful for the contents of dynamic web page extracted from RDB. Their Framework, DPAnnotator, translates the ER schema of the RDB into OWL ontology. They provide a D2OMapper tool, which automatically creates the mappings by following their rules; their approach and tool can act as a gap-bridge between existing database applications and the semantic web. Another simplistic platforms approach is Triplify offered by Auer et al. [24] for publishing RDB as RDF graph. The created RDF graph in this method can be either published as Linked Data or materialized, thus allowing dynamic access. The drawback of this tool is a necessity to manually write the SQL commands for generating an RDF triple. Moreover, the W3C RDB2RDF Working Group recommended a standard customized mapping language, for expressing RDB-to-RDF mappings document manually, called R2RML [25]. This approach requires an expert for complete mapping of RDB to the existing ontology, particularly to avoid problems that occur during mappings document constraints. Recently, M. A. Hazber et al. [26] presented tool to produce an R2RML mappings document automatically from an RDB schema according to the direct mapping specification [27]. Thus tool supports any R2RML engine of generating RDF triples accessing RDB data and producing a set of RDF dataset.

2.3. Semantic Query in RDB Using Ontology

The mapping between RDB and Semantic Web ontology is not enough to integrate a relational database into the Semantic Web, thus query processing semantically over these mappings is required. Expressing SPARQL queries against graph structured ontology in the SQL query language is one of the fundamental problems for the development of the Semantic Web. However, these approaches have some drawbacks especially in integration with SQL queries and in conversion of SQL query to the corresponding language data format. RDF query language (SPARQL) [6], [7] presents a standard language for querying on RDF data that focuses on the transformation of traditional SQL queries to RDF query languages. D2R Server [28] is an engine to publish the RDB content as RDF graph. It uses D2RQ mappings for translating query requests from external applications to SQL queries on the RDB. The methodology, proposed by Banu et al. [29] based on Library Management System (LMS) database to build ontology. This method depends on two steps, ontology extraction (offline) and semantic query (online). In the offline, the process extracts the ontology from explicit relations of LMS schema and then the domain expert will modify the ontology by adding the implicit relations to complete generated ontology. In online query operation, the user can submit a request of semantic query to the system, and the system translates that query into a related SQL query for the underlining RDB. To extract ontology from RDB, two rules are applied on the primary key and foreign key, both rules are non-final, because not all object properties are defined. The disadvantage of this approach needs domain experts to complete generating ontology. Lee and Sohn [30] proposed a framework, which can automatically create ontology from a relational schema and can clearly discover the semantic relations between data through the ontology building process. The framework proposed in this approach consists of two modules, MOG (Module for Ontology Generation) and Module for MQO (Query using the Ontology). MOG is a module that generates the ontology from RDB schema, whilst MQO is a supporting module for executing the query using the ontology. Ranganathan and Liu [31] specified three types of semantically relevant results, which are direct, inferred, and related results. These types are based on their relationship to the semantic query and how these results can be obtained. Based on ontology models, the end-user can expresses the semantic queries and those queries are translated into a syntactic SQL queries. The semantic

queries are based on SPARQL where the user can issue either schema or data query. Rodriguez-Muro et al. [32] proposed the ontop system that allows SPARQL queries over RDF views of RDBs. They converted SPARQL query to datalog programs and then rewritten and converted to the SQL query. Cyganiak [33] described the transformation of SPARQL language to relational algebra and outlines a set of transformation rules to create the equivalence between algebra and SQL. The methodology of their approach depends on a global reference table that contains RDF statements in form (subject, property, object) as a basis for the operations. The drawback of this approach lacks the nested OPTIONAL pattern problem.

Compared with existing approaches, this work is quite different in terms of an integrated method, since we added new complementary concepts to analyze RDB and SWO. These ideas then used to ensure further analysis reflecting the integration of our work. For example, we produce ontology schema from RDB schema, transform the contents of RDB (considering null-values) to RDF triples, and re-writing SPARQL queries corresponding to relational algebra to easy enable the web applications to queries on RDB as RDF triples. The strength of this work it takes account of the important concepts of RDB, such as constraints, relationships, and null-values for all phases of the transformation rules that are demonstrated with examples. The validation, results, and implementation are also considered.

3. Preliminaries

This section mainly presents a good notation and definitions used in this paper. It defines the basic terminology of RDBs and Semantic Web ontology languages in order to make the rules of this work understood and applicable.

3.1. Relational Databases

A Graph Data Model for Relational Databases: Typically, we assume that: (i) a relational database schema $RDBS$ is a set of relation schemas $TB_1(A_1),..,TB_n(A_n)$, where TB_i is the name of the $i\text{-}th$ relation (table) and A_i is the set of its attributes (columns) name that denoted by $att(TB_i)=\{A_1,..,A_n\}$, and (ii) a relational database RDB over $RDBS$ is a set of instance (data) of relations $I_1,...,I_n$ over $TB_1(A_1),..,TB_n(A_n)$, respectively. Where I_i is a set of tuples (rows) $rw_1,...,rw_m$ over TB_i that denoted by $I(TB_i)=\{rw_1,...,rw_m\}\in RDB\in RDBS$ for all attributes in TB_i, where each row $rw_i:1\le i\le m, and\ \ I_i(TB_i(A_i))=\{rw_1.A_i,...,rw_m.A_i\}\in RDB\in RDBS$ for A_i in TB_i. The notation $rw_i.A_i$ (or $val(A_i,rw_i,TB_i)$) refers to the value of a row rw_i in a column A_i, (iii) each column A_i has a data type $A_{i(\text{type})}$ ($type$ i.e. string, int, float, date, etc.). In the following, we underline the attributes of a relation that belong to its primary key $pk_1(A_1,TB_i)$ and we denote for foreign key by $TB_i.A \overset{fk}{\rightarrow} TB_k.B$ (or $fk(A,TB_i,B,TB_k)$) between the attribute A of a relation TB_i and the attribute B of a relation TB_k.

Basic Structure of SQL Queries: The fundamental structure of an SQL expression depends on three clauses: **SELECT**, **FROM**, and **WHERE** clause, which they are corresponding to the **PROJECTION**, **CARTESIAN-PRODUCT**, and **SELECTION PREDICATE** operations of the relational algebra respectively. A typical SQL query has the following form:

Definition 1: **SQL query**	
Select $A_1,..,A_m$ **From** $TB_i,..,TB_n$ **Where** P	P *is a predicate*

Relational Algebra (Q): We use relational algebra as a query language for relational data. The important basic operation of relational algebra: delete unwanted attributes **PROJECTION**($\prod$), select tuples **SELECTION**(σ), combine relations **JOIN**(∞), set operations **UNION**($\cup$) and **DIFFERENCE**($\setminus$), and **RENAME**(ρ) are defined in our work. The Definition (1) can be re-written into equivalent a relational

algebra as follows:

Definition 2: **SQL algebra** (Q)	
$\prod_{A_1,...,A_m,\sigma_P}(TB_i \times .. \times TB_n)$	$n,m \geq 0$ p stand for a condition like ($\sigma_{A_i=x}$): x is variable or value.
$A_1,...,A_m \in att(TB_i \times .. \times TB_n) \subseteq \text{att}(RDB(TB_1,..,TB_n)) \in RDBS$	

3.2. Semantic Web Ontology Languages

A Graph Data Model for Ontology: Typically, we assume that: (i) a semantic web ontology schema *SWOS* is a set of classes (owl:class) $CLS_1(DTP_1),..,CLS_n(DTP_n)$, where CLS_i is the name of the *i-th* class and DTP_i is the set of its datatype properties names (owl:Datatype Property) to describe the properties of classes that denoted by $att(CLS_i) = \{DTP_1,..,DTP_n\} \in CLS_i$ and (ii) a semantic web ontology *SWO* over *SWOS* is a set of instance of class $Io_1,...,Io_n$ over $CLS_1(DTP_1),..,CLS_n(DTP_n)$, respectively. Where Io_i is a set of tuples (graph) $tr_1,...,tr_m$ over CLS_i that denoted by $Io(CLS_i) = \{tr_1,...,tr_m\} \in SWO \in SWOS$ for all datatype properties in CLS_i, where each triple $tr_i : 1 \leq i \leq m$, and $Io_i(CLS_i(DTP_i)) = \{tr_1.DTP_i,...,tr_m.DTP_i\} \in SWO \in SWOS$ for DTP_i in CLS_i. The notation $tr_i.DTP_i$ (or $val(DTP_i,tr_i,CLS_i)$) refers to the value of a tuple (triple) tr_i in an property (attribute) DTP_i, (iii) each (*owl:ObjectProperty*) OBP_i and DTP_i has a set of domain (rdfs:domain) Dom and range (rdfs:range) Rng classes, and DTP_i has a XML schema data type $DTP_{i(xsd)}$ [34] (e.g. *rdf:resource="xsd;string","xsd;int","xsd:float","xsd;date",etc.).* Each class has a set of object properties OBP_i to describe the relations between properties of classes $CLS_i(DTP_i)$ denoted by $CLS_{i(Dom)}(DTP_i) \rightarrow^{(OBP_i)} CLS_{k(Rng)}(DTP_k)$ *or* $OBP_i(DTP_i, CLS_{i(Dom)}, DTP_k, CLS_{k(Rng)})$ that mean OBP_i is a relationship between $CLS_{i(Dom)}(DTP_i)$ *and* $CLS_{k(Rng)}(DTP_k)$ through its domain Dom and range Rng.

RDF Graph (Triples) and OWL Vocabulary: Assume there are pairwise disjoint infinite sets IR(*IRIs*) to denote web resources, BN (blank nodes) to denote a special type of objects that describe *anonymous resources* and L (literals) denotes the values (e.g. Natural Numbers, Boolean, Date Time, and String). A tuple $(s,p,o) \in (IR \cup BN) \times IR \times (IR \cup BN \cup L)$ is called an RDF triple $t = (s,p,o)$, where s is the subject, p is the predicate and o is the object as shown in Fig. 1. A finite set of RDF triples is called an RDF graph $G = \{t_1,...,t_n\}$. Moreover, assume the existence of an infinite set V of variables disjoint from the above sets, and assume that every element in V starts with the symbol ? such as $?V_1,...,?V_n$.

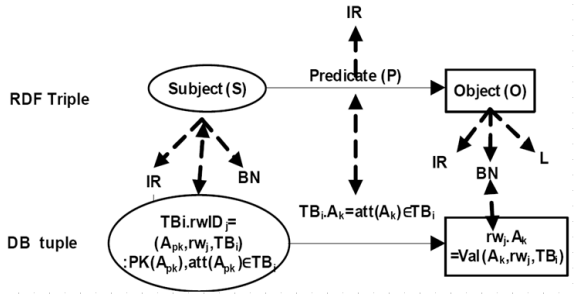


Fig. 1. RDF triple corresponds to the record tuple in RDB.

SPARQL Query (Q°): Semantic queries for the semantic web data, represented by RDF graphs, are specified using some of the W3C standard query languages, e.g. SPARQL. In this paper, we use SPARQL as a query language for accessing RDF graphs (triples). The official syntax of SPARQL [6], [7] consider these operators **SELECT**, **FILTER**, **OPT**(OPTIONAL), **UNION**, **AS** and concatenation via a dot symbol • (**AND**), which

denote the end of a triple pattern TP, to construct graph pattern GP expressions. A set of graph patterns is a group graph pattern GGP (or {}), meaning that each part of group graph pattern must be matched. A SPARQL uses a **WHERE** clause to define *graph patterns* to discover a match for in the query data set. In SPARQL queries, variable names $V:V_1,..,V_n$ are prefixed with the question mark ? symbol e.g. $?V:?V_1,...,?V_n$, and value constraints can be defined by the FILTER keywords. The SPARQL query offers lots of filtering possibilities: String matching to test strings based on regular expressions **regex** $(?V,str)$, Boolean operators $(!,||,\&\&)$, comparison operators $(=,!=,>,>=,<,<=)$, arithmetic operators $(*,/,-,+)$, and RDF element operators **bound**$(?V)$, **isURI**$(?V)$ and **STR**(). Precisely, a SPARQL graph pattern expression GP is defined as the following abstract grammar:

Grammar (GP) : **SPARQL graph pattern expression**
$GP \rightarrow TP \mid GP_i$ **AND** $GP_k \mid GP_i$ **OPT** $GP_k \mid GP_i$ **UNION** $GP_k \mid GP$ **FILTER** $expr\_cond$
$TP \in (IR \cup BN \cup L \cup V) \times (IR \cup V) \times (IR \cup L \cup V)$

where, **FILTER** $expr\_cond$ represents the FILTER construct with a Boolean expression $expr\_cond$. Therefore, a SPARQL query defined as:

Definition 3: SPARQL query (Q°)	
$Q^{\circ} =$ **SELECT** $?V_1,...,?V_n$ **WHERE** $\{GP$ **FILTER** $(expr\_cond)\}$	$?V_1,...,?V_n \in var(GP)$

For example, a SPARQL graph pattern GP can be represented as:

$$(((?X, name, ?N) \textbf{AND } (?X, age, ?Ag)) \textbf{ FILTER}(\textbf{regex}(?N), "mohamed")).$$

4. Rules for Generating Ontology from Relational Database

In this section, we introduce the first part of our approach to construct OWL/RDF triples from RDB, automatically by using a set of particular rules, called mapping rules. This method presents the basic transformation rules for producing RDF triples from relational data including null values, and enables semantic query engines to answer more semantic queries. The contents of this part are represented by running examples, which includes the important cases, such as relationships of RDB as shown in Fig. 2.

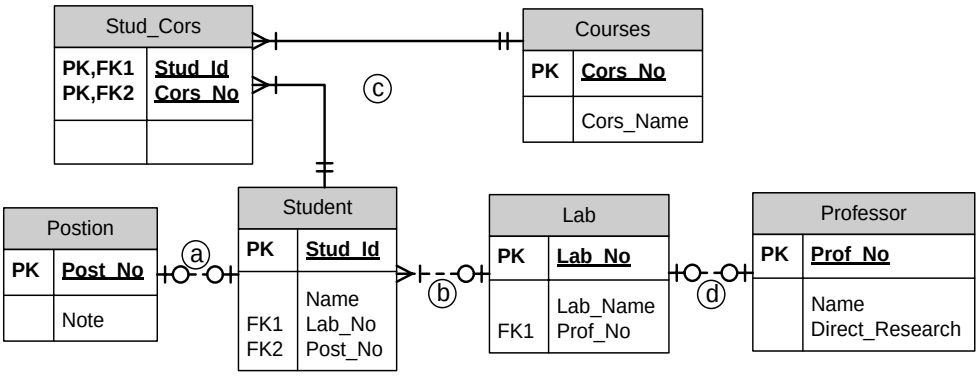


Fig. 2. Relationships and constraints in RDB laboratory (RDBLAB).

4.1. Rules for Mapping a RDBS to Ontology

Now we will define the rules that map 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 (Definition 4) and Identify Binary Relation (Definition 5):***

Definition 4: **IsRelationship**($TB_i.A, TB_k.B$)	**Definition 5:** **ISBinaryRel**(TB_b)
$TB_i(A_1,\ldots,A_n), TB_k(B_1,\ldots,B_n), TB_i.A \rightarrow^{fk} TB_k.B$ $\rightarrow$ **HasRelationship**(TB_i, A, TB_k, B) $\rightarrow$ **IsRelationship**($TB_i.A, TB_k.B$)	**IsRelationship**($TB_b.A_1, TB_i.B$), **IsRelationship**($TB_b.A_2, TB_k.C$), $pk_2(A_1, A_2, TB_b)$, $pk_1(B, TB_i)$, $pk_1(C, TB_k)$? $\rightarrow$ **BinaryRel**($TB_b, A_1, A_2, TB_i, B, TB_k, C$) $\rightarrow$ **ISBinaryRel**(TB_b)
i, k are name of tables and $A \in att(TB_i), B \in att(TB_k), TB_i, TB_k \in TB \subseteq RDBS$	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 following rules:

Rules for Non-binary Relation: Every concept in RDBS that belongs to a non-binary relation is mapped to ontology according to the following rules:

Rule1 ($TB_i \rightarrow CLS_i$): Each table TB_i in an RDB unless **ISBinarRel**(TB_i) should be mapped to a class CLS_i in SWO.

Rule1(R_1): $TB_i \rightarrow CLS_i$		
RDBS	*SWOS* (RDF/OWL)	
$TB_i \wedge$ **!ISBinaryRel**(TB_i)	CLS_i	i is name, $TB_i \in RDB, CLS_i \in SWO$.

This is a basic rule used to identify classes in a clear way from several cases in tables. It depends on the function condition, ISBinaryRel(TB_i) in Formula (5). Such cases include:

- All tables that have default attributes with (or without) primary keys in absence of foreign keys should be mapped to classes. For instance, tables (Postion, Professor, and Courses) are mapped to classes (Postion, Professor, and Courses).
- All tables that have only one or more than two FKs should be mapped to classes. For instance, tables (Lab, Student) should be mapped to classes (Lab, Student).
- All tables that have two FKs with one or more non-FK attributes should be mapped to classes.
- All tables that have two FKs(A,B) but not PKs(A,B) should be mapped to classes.

Therefore, Rule1 identify all the classes (Postion, Professor, Courses, Lab, and Student) from our RDBLAB schema except the table (Stud_Cors) are not mapped, because the condition (!ISBinaryRel("Stud_ Cors") = false).

Rule2 ($TB_i(A_{k(type)}) \rightarrow CLS_i(DTP_{k(xsd)})$): Each column $A_k \in TB_i$ unless $pk(A_k, TB_i)$ or $fk(A_k, TB_i)$ and **!ISBinaryRel**(TB_i) should be mapped to a datatype property DTP_k in class $CLS_i \in SWO$ and datatype of column (type) to XML schema datatype (xsd) of property.

Rule2 (R_2): $TB_i(A_{k(type)}) \rightarrow CLS_i(DTP_{k(xsd)})$	
RDBS	*SWOS* (RDF/OWL)
$TB_i(A_{k(type)}) \wedge !(pk(A_k, TB_i) \vee fk(A_k, TB_i)) \wedge$ **!ISBinaryRel**(TB_i)	$CLS_i(DTP_{k(xsd)})$

For identification of datatype properties to make a relationship between instances of classes with RDF literals and XSD, Rule 2 was used. It covers several cases for mapping columns of table to the datatype properties (DTPs). This rule depends on the predicate expressions $TB_i(A_{k(type)})$, $pk(A_k, TB_i)$, $fk(A_k, TB_i)$, and ISBinaryRel(TB_i). The cases can be described as follows:

- All default columns (not PKs or FKs) that have datatype should be mapped to the datatype property with domain and range (xds datatype corresponding to SQL datatype).
- All table columns that are ISBinaryRel(TB_i)=true are not mapped to datatype properties.
- Every column unless the predicates $pk(A_k, TB_i)$= true and $fk(A_k, TB_i)$=true should be mapped to datatype properties with their xsd datatype according the columns SQL datatype.

For instance, the default column (Lab_Name) in table (Lab) is hold in our example. The obtained result is

owl:DatatypeProperty (Lab_Name) with domain (Lab class) and range (xsd^^string corresponding the original SQL datatype of column (Lab_Name$_{\textbf{varchar}}$)).

Rule3 ($pk(A_k,TB_i) \rightarrow INVFUNPR_k(A_k)$): Each attribute $A_k \in TB_i$ that is $pk(A_k,TB_i)$ unless $fk(A_k,TB_i)$ should be mapped to both an inverse functional property $INVFUNPR_k(A_k)$ (owl:InverseFunctionProperty) with a minimum cardinality of 1 (*owl:minCradinality*) $minCRD1(A_k)$ restriction on the property A_k.

Rule3 (R_3): $pk(A_k,TB_i) \rightarrow INVFUNPR_k(A_k)$	
RDBS	***SWOS* (RDF/OWL)**
$pk(A_{k(type)},TB_i) \wedge !fk(A_{k(type)},TB_i) \wedge$ **!ISBinaryRel**(TB_i)	$INVFUNPR_k(A_k,CLS_{i(Dom)},A_{k(xsd)\text{-}Rng})$, $minCRD1(A_{k(xsd)})$

This rule used for mapping column primary key to inverse function property $INVFUNPR_k(A_k)$ and restriction minCardinality constraint of 1 $minCRD1(A_k)$, if the predicate conditions of rules are true. Two cases, unique and not null column properties can be inferred by implicit way, since the primary key is a column that contains a unique and not null value for each row in the table. Therefore, if the column is a primary key, it should be mapped to $INVFUNPR_k(A_k)$ **(**unique constraint**)** and to restriction $minCRD1(A_k)$ (not null constraint**)**. For instance, the primary key Lab_No in table Lab is hold in our example, where pk(Lab_No$_{\text{int}}$) ∧ !fk(Lab_No$_{\text{int}}$) ∧ !ISBinaryKey(Lab) are true, then INVFUNPR$_{\text{Lab_No}}$ (Lab_No,Lab$_{\text{domain}}$,xsd^^int) and minCRD 1(Lab_No) will be generated.

Rules for Relationships **between Tables**: Relationships in RDBs are maintained through the use of foreign keys. A foreign key is the basis of any relationship between relations $TB_1,...,TB_n \in RDB$. Therefore, we will start to map it based on our analysis in Section 3.1.

Rule4 ($TB_i.A \rightarrow^{(fk)} TB_k.B$): Each attribute (column) A in the table TB_i that references attribute B in the table TB_k should be mapped to an object property (owl:ObjectProperty) OBP that has the source table TB_i as its domain ($TB_i \rightarrow CLS_{i(Dom)}$) and destination table TB_k as its range ($TB_k \rightarrow CLS_{k(Rng)}$).

Rule4 (R_4): $TB_i.A \rightarrow^{(fk)} TB_k.B$	
RDBS	***SWOS* (RDF/OWL)**
$fk(A,TB_i,B,TB_k)$	$OBP(A,CLS_{i(Dom)},CLS_{k(Rng)})$

Rule 4 is the basic rule for transforming the relationship between two tables through foreign keys to object properties. Object property can be defined as a relationship between instances of two classes through a domain and range.

There are three types of relationships in a relational database, one:one(or zero) (1:1/0), one:many (1:m), and many:many (n:m), as shown in Fig. 2, which will be mapped to ontology according to the following rules:

Rule5.1 ($TB_i.A \rightarrow^{(one:one)} TB_k.B$ **):** If two relations $TB_i(A_1,...,A_n)$ and $TB_k(B_1,..,B_n)$ are related to each other through their columns $TB_i.A \rightarrow^{(fk)} TB_k.B$, where $fk(A,TB_i)$ references $pk_1(B,TB_k)$, the relation should be 1:1 ($TB_i.A \rightarrow^{(one:one)} TB_k.B$). Therefore, $fk(A,TB_i)$ is mapped into $OBP(A,CLS_{i(Dom)},CLS_{k(Rng)})$ that has the source table TB_i as its domain, and the destination table TB_k as its range. In the 1:1 relationship, the value $TB_i.A = TB_k.B$. The property is restricted to the same value from the class (TB_k) $RestOnProp(A,hasValue,TB_k)$. The constraint $A \neq null$ is mapped into a minCardinality in the same restriction, and the property (A) *RestOnProp(<A,owl: hasValue,TB$_k$>,<A,owl:minCardinality,xsd^^int 1>).* Fig. 2(a) illustrates an example of this case where only one position in a laboratory (Lab) holds by one student.

Rule5.1(R_5.1): $TB_i.A \rightarrow^{(one:one)} TB_k.B$	
RDBS	***SWOS* (RDF/OWL)**

IsRelationship$(TB_i.A, TB_k.B) \wedge pk_1(B, TB_k) \wedge A \neq null$	$OBP(A, CLS_{i(Dom)}, CLS_{k(Rng)})$, $RestOnProp(< A, owl:hasValue, TB_k >, A, owl:minCardinality, xsd\text{^^}int\ 1>)$

This rule reflects to one:one relationship between two classes in ontology through object properties as shown in Fig. 2(a). By applying this rule, the predicate conditions IsRelationship (Student.Post_No, Postion.Postion.Post_N $\wedge$ pk_1(Post_No,Postion) $\wedge$ Student.Post_No≠ null are true. Hence, the OBP(Post_No, $Student_{doman}$,$Postion_{rang}$), RestOnProp(<Post_No,owl:hasValue,Postion>,<Post_No,owl:minCardinality, xsd^^int 1>) are generated. Therefore, the ontology can be extracted practically as shown in Fig. 3.

Rule5.2 ($TB_i.A \rightarrow^{(one:many)} TB_k.B$ **):** If the function **IsRelationship**$(TB_i.A, TB_k.B)$ is true, and $TB_i.A \rightarrow^{(one:one)} TB_k.B$ is false, the relationship is 1: m, and $TB_i.A \rightarrow^{(one:many)} TB_k.B$ is mapped into an $OBP(A, CLS_{i(Dom)}, CLS_{k(Rng)})$. In the 1: m relationship, a column value ($TB_k.B$) exists in the column value ($TB_i.A$). Therefore, this property is restricted to all values from the class (TB_k) $RestOnProp(A, owl:allValueFrom, TB_k)$. If the constraint $A \neq null$ holds, it will be mapped into a mincardinality $RestOnProp(< A, owl:allValueFrom, TB_k >, < A, owl:minCardinality, xsd\text{^^}int\ 1>)$. This case can be illustrated in Fig. 2(b). The following rule is used for extracting ontology for object properties and restriction, when the foreign key represents a relationship as 1: m.

Rule5.2(R_5.2) : $TB_i.A \rightarrow^{(one:many)} TB_k.B$	
RDBS	***SWOS* (RDF/OWL)**
IsRelationship$(TB_i.A, TB_k.B) \wedge$!(**ISBinarRel**$(TB_i) \vee$ **ISBinarRel**$(TB_k)) \wedge$!$(TB_i.A \rightarrow^{(one:one)} TB_k.B) \wedge pk_1(B, TB_k) \wedge valOf(A, TB_i, From, B, TB_k) \wedge A \neq null$	$OBP(A, CLS_{i(Dom)}, CLS_{k(Rng)})$, $RestOnProp(< A, owl:allValueFrom, TB_k >, < A, owl:minCardinality, xsd\text{^^}int\ 1>)$

This rule reflects 1:m relationship between two classes in ontology through object properties. It also contains the same predicate conditions in R_5.1 with additional predicate $valOf(A, TB_i, From, B, TB_k)$, to ensure all the values of column TB_i.A are from TB_k.B. The generated ontology has the restriction $RestOnProp(< A, owl:allValueFrom, TB_k >)$ indicating all the values property (A) are from the class (TB_k). For example (Fig. 4), the relationship between Student and Lab are holds. The Lab_No is a foreign key in the table Student that references column Lab_No in the table Lab. A student studies in one Lab, and any given lab has one or more students studying there.

```
<owl:ObjectProperty rdf:ID="Studen.Post_No">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Postion"/>
 </owl:ObjectProperty>
<owl:Class rdf:about="#Student">
  <rdfs:subClassOf>  <owl:Restriction>
    <owl:onProperty rdf:resource="#Studen.Post_No"/>
    <owl:hasValue  rdf:resource="#Postion"/>
   <owl:minCardinality rdf:datatype="&xsd;int"1/> [Delete IF 1:0]
      …
</owl:Class>
```

Fig. 3. OWL extracted from relationship (1:1) between tables Student and Position

```
<owl:ObjectProperty rdf:ID="Studen.Lab_No">
      <rdfs:domain rdf:resource="#Student"/>
      <rdfs:range rdf:resource="#Lab"/>
 </owl:ObjectProperty>
<owl:Class rdf:about="#Student">
        ….
       <owl:onProperty rdf:resource="#Studen.Lab_No"/>
        <owl:allValueFrom  rdf:resource="#Lab"/>
         <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger"1/>
    ….
</owl:Class>
```

Fig. 4. OWL extracted from relationship (1:m) between tables Student and Lab

Rule5.3 ($TB_i.A \leftarrow^{(many\ \ A.TB_{b.}B^{many})} \rightarrow TB_k.B$ **):** In the relationship n:m the maximum of both multiplicities is greater than one, for example, the assigned relationship between Student and Course. A student is assigned one or more courses, and each course is assigned to one or more students. If two tables $TB_i(A_1,..,A_n)$ and $TB_k(B_1,...,B_n)$, are related to each other through the third table $TB_b(A,B)$ where $pk_2(A,B,TB_b)$, $TB_b.A \rightarrow^{fk} pk_1(A,TB_i)$ and $TB_b.B \rightarrow^{fk} pk_1(B,TB_k)$ then **BinaryRel**$(TB_b, A, B, TB_i, A, TB_k, B)$ is hold. In such a situation, only the tables $TB_i(A_1,..,A_n)$ and $TB_k(B_1,..,B_n)$ are represented in the ontology as classes with two object properties and their restrictions. Therefore, the binary relation is mapped into two $OBP(A, CLS_{i(Dom)}, CLS_{k(Rng)})$ and $OBP(B, CLS_{k(Dom)}, CLS_{i(Rng)})$ according to the rule of (R_5.2) 1:m relationship.

Rule5.3 (R_5.3): $TB_i.A \leftarrow^{(many\ \ many)}_{A.TB_{b.}B} \rightarrow TB_k.B$	
RDBS	***SWOS* (RDF/OWL)**
ThreeTables$(TB_i(A_1,..,A_n),$ $TB_k(B_1,..,B_n), TB_b(A,B)) \wedge$ **ISBinarRel**(TB_b)	callRuleMap$(TB_i.A \rightarrow^{(one:many)} TB_k.B)$, callRuleMap$(TB_k.B \rightarrow^{(one:many)} TB_i.A)$

The relationship between two classes through their object properties can be represented by Rule 5.3. The major property of this rule is that it has ability to call Rule 5.2 twice in reversible directions. First, it calls Rule5.2 $(TB_i.A \rightarrow^{(one:many)} TB_k.B)$ to map the relationship one:many between TB_i and TB_k. Secondly, it calls Rule5.2 $(TB_k.B \rightarrow^{(one:many)} TB_i.A)$ into reversal direction. According to the Formula (5), **BinaryRel**(Stud_Cors, Stud_Id,Cors_No,Student,Stud_Id, Courses, Cors_No) holds in our example in Fig. 2(c). The table Stud_Cors has two columns. (Stud_Id, Cors_No) is the primary key of Stud_Cors, Stud_Id is a foreign key in Stud_Cors that references column Stud_Id in Student, and Cors_No is a foreign key in Stud_Cors that reference column Cors_No in Course. Therefore, the binary relation in Fig. 2(c) is mapped according to the above rule.

Generating IRI for the Triples of the Schema: During the mapping process, a prefix (Name space) IRI denoted by NS_{IRI} for the RDB should also be translated (e.g. NS_{IRI} :http://mo_exp.edu.cn/dbLab/#). The following rules to produces IRIs are:

Rule to generate IRI for the class:	
Input	***Output***
$(NS_{IRI} + CLS_i)$	$CLS_{i(IRI)}$

Rule to produce IRI for the any property in the class:	
Input	***Output***
$CLS_{i(IRI)} + "." + A : A \in att(CLS_i)$	$CLS_i(A)_{(IRI)}$

Example: ("http://mo_exp.edu.cn/dbLab/#Professor",rdf:type,owl:Class) where the (http://mo_ exp.edu.cn/dbLab/#Professor) is the IRI for the Professor table in our example. Another example, the triple("http://mo_exp.edu.cn/dbLab/#**Professor.Name"**,rdf:type,owl:datatypeProperty), and the triple(http://mo_exp.edu.cn/dbLab/#Student.Lab_No,rdf:type,owl:objecrProperty) are hold. This rule helping to avoid confusion, providing clear triples of ontology schema, and indicating the source of relational schema. It can also prevent the names of properties from being duplicated even though two or more tables have the same name of a column. For example, the tables Student and Professor have the same name of column "Name". After applying our rules, the generated ontology properties are **Student.Name** and **Professor.Name**.

4.2. Rules for Generating RDF Triples from RDB Instances

We define the rules that map a RDB instance into RDF triples, in order to establish simple way and data loss avoidance; moreover, to access RDF triples using semantic search technologies. If a table TB_i is mapped to the class $CLS_{i(IRI)}$ then all rows of the table $I(TB_i) = \{rw_1,...,rw_n\}$ are transformed to the instance of RDF graphs $Io(CLS_i)_{(IRI)} = \{gr_1,...,gr_n\} : gr_i = \{t_1,...,t_{nc}\}, t = (s,p,o)$. If each column of table $TB_i(A_1,...,A_n)$ transferred to the properties of class $CLS_{i(IRI)}(DTP_{1(IRI)},...,DTP_{n(IRI)})$, then the values of the columns unless null-value in table $I(TB_i) = \{rw_{1(A_1,...,A_{nc})},...,rw_{n(A_1,...,A_{nc})}\}, val(A_i, rw_k, TB_i) \neq \text{null}$ can be mapped to the values of the corresponding property of ontological instance $Io(CLS_i)_{(IRI)} = \{gr_1.t_{(A_1,...,A_{nc})},...,gr_n.t_{(A_1,...,A_{nc})}\}$. Firstly, we initiate a family of predicates that produce $\mathbf{RwId_{k(IRI)}}$ for the triples being translated according to the following rule.

Definition (6): Procedure for generating $\mathbf{RwId_{k(IRI)}}$	
Function header	$RowIRI(\mathbf{RwId_{k(IRI)}}, TB_i, rw_k)$

Function body Definition	$TB_i(A_1,..,A_{nc}), pk_n([A_i,...,A_{nc}],TB_i), val(A_1,rw_k,TB_i),...,val(A_{nc},rw_k,TB_i),$ $collect\text{-}to\text{-}RwId(NS,TB_i,"\_",collect(val(A_1),...,val(A_{nc}),\mathbf{RwId_{k(IRI)}})$

A $RowIRI(\mathbf{RwId_{k(IRI)}},TB_i,rw_k)$ generates the identifier $\mathbf{RwId_{k(IRI)}}$ of a row rw_k of a relation TB_i. Thus, given that the facts PK1("Student","Stud_Id") and VALUE("Student","rw1","Stud_Id",1) are hold in our example, the (RwId1=NS:Student_1) is the identifier for the tuple in table Student with value 1 in the *pk*. To generate triples for data row columns of the table our following rule generated the RDF triples from RDB instance.

Rule6: $I(TB_i)_{(rw1,...,rw_n)} \rightarrow Io(CLS_i)_{(gr_1,...,gr_n)}$.

Procedure for generating RDF datasets from RDB data.

$I(TB_i)_{(rw1,...,rw_n)}$	$Io(CLS_i)_{(gr_1,...,gr_n)}$.
$rw_r([A_1,...,A_{nc}],TB_i)$ $\wedge\ val([A_1,...,A_{nc}],$ $rw_r,TB_i) \neq$ null	$RowIRI(\mathbf{RwId}_{r(\mathbf{IRI})},TB_i,rw_r),\mathrm{gr}_r$. $\mathrm{gr}_r =$ $\begin{pmatrix} t_{id}(\mathbf{RwId}_{r(\mathbf{IRI})},rdf{:}type,CLS_{i(IRI)}), \\ t_1(\mathbf{RwId}_{r(\mathbf{IRI})},CLS_i(A_1)_{IRI)},\mathrm{getObject}(A_1,rw_r,TB_i)),...., \\ t_{nc}(\mathbf{RwId}_{r(\mathbf{IRI})},CLS_i(A_{nc})_{IRI)},\mathrm{getObject}(A_{nc},rw_r,TB_i) \end{pmatrix}$.

Predicate function of $\mathbf{getObject}(A_k,rw_r,TB_i)$

Function header	$\mathbf{getObject}(A_k,rw_r,TB_i)$
Function body Definition	*if* $(!\mathrm{Isfk}(A_k,TB_i) \wedge val(A_k,rw_r,TB_i) \neq null)$ return$(val(A_k,rw_r,TB_i))$; *elseif* $(\mathbf{Isfk}(A_k,TB_i),\mathrm{fk}(A_k,TB_i,B,TB_c))$ return$(RowIRI(\mathbf{RwId}_{rc(\mathbf{IRI})},TB_c,rw_{rc}))$;

Where, $\mathbf{Isfk}(A,TB_i) \leftarrow [(TB_i.A \rightarrow^{(fk)} TB_i.B), val(A_k,rw_r,TB_i) \neq null]$. Note that, this rule expresses the conversion of the null-value and its retrieval in query is explained in Section 5. The function $\mathrm{getObject}(A_k,rw_r,TB_i)$ used to get the object of the triple t_k of graph gr_r depends on the type of column A_k (literal or foreign column). The table triples in our example, (as shown in Fig. 5) is:

Let xmlns:NS="http://www.mo_exp.edu.cn/dbLab/#".

$Student(gr_1(t_{id}(\mathrm{NS:Student\_1},\ \mathrm{rdf:type},\ \mathrm{NS:Student}),\ t_1(\mathrm{NS:Student\_1},\ \mathrm{NS:Student.Name},\ \mathrm{Mohamed}),\ t_2(\mathrm{NS:Student\_1},\ \mathrm{NS:Student.Lab\_No},\ \mathrm{NS:Lab\_442}),...),...,).$

$Lab(gr_1(t_{id}(\mathrm{NS:Lab\_442},\ \mathrm{rdf:type},\ \mathrm{NS:Lab}),\ t_1(\mathrm{NS:Lab\_442},\ \mathrm{NS:Lab.Lab\_No},\ 442),...),...,).$

Therefore, for understanding how to apply our rules on RDBs to generate ontology schema and RDF graph (triples), the following example (Fig. 5) is used: If one of the tables refers to another table by a foreign key, it can be mapped according to the above rules R_1 (table to class), R_2 (column with datatype to datatype property with XML schema data type), R_3 (primary key to inverseFunctionProperty), R_4 and R_5 (relationships to object property with restriction on property), and R_6 (rows to triples). From the instances mentioned in Fig. 5, it can be observed that the values of rows in the table Stud_Cors are not simply represented as literals instead of properties added to the classes of Student and Courses. These properties link the resources between Student and Course nodes in the RDF/OWL, also the values of Lab_No in a Student represented as a property added to the class of Student because table a Student has a column Lab_No that references a table Lab through a column Lab_No. Therefore, a created class Student has one property linking the resources Lab node to represent the values of a column Lab_No in the Student.

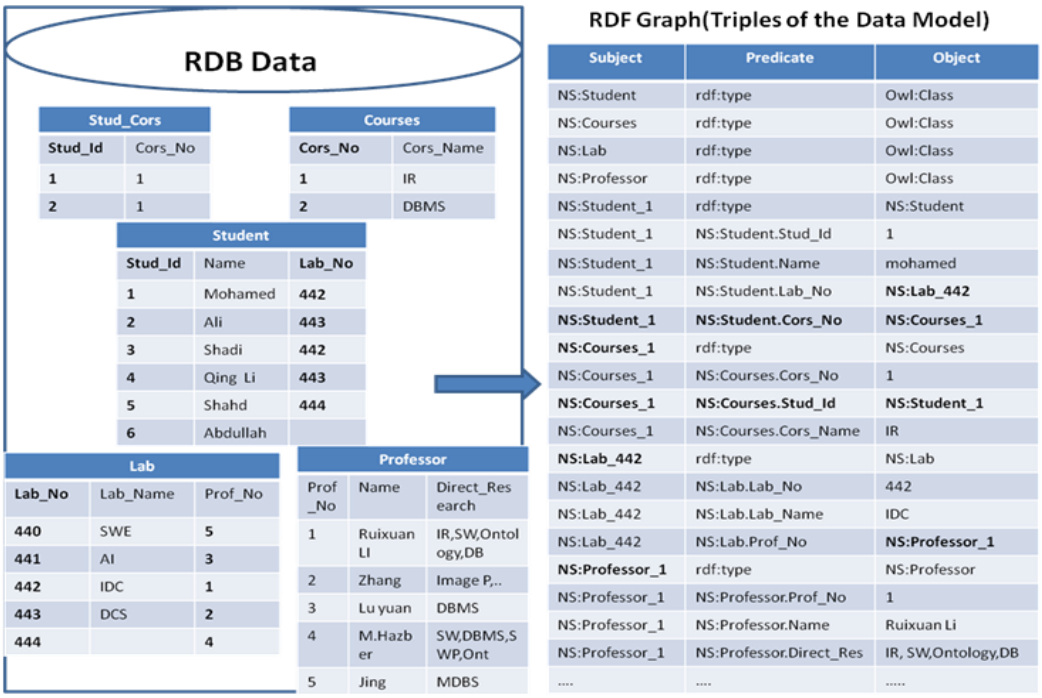


Fig. 5. Part of RDF triples of the data model extracted from the RDB data of relationships (1:1, 1:M and N:M).

5. Rules for Generation of SPARQL Query from SQL Algebra

The result of the previous section is an RDF graph. It is created in an automatic transformation mechanism from the data stored in RDBs, which can be processed by most of Semantic Web. Therefore, Semantic Web applications need to be accessing relational database contents by semantic methods. Presently, SPARQL is a W3C recommendation and has been becoming the standard language for querying RDF data. All queries presented in this paper have been verified using Apache Jena-ARQ implementation of SPARQL. Assume a given relational instance I over TB (possibly including null values). We proof and explain that, for every relational algebra query $Q(I(TB))$, there is SPARQL query $Q^{o}(Io(CLS))$ satisfying the following function:

Definition (7): $Q(I(TB)) \rightarrow Q^{\circ}(Io(CLS))$	
SQL- $Q(I(TB))$	SPARQL- $Q^{\circ}(Io(CLS))$
$\{rw_1,...,rw_n\}.$ $rw_i = \{rw_i.A_1,...,rw_i.A_{nc}\}.$	$\{gr_1,...,gr_n\}.$ $gr_i = \{t_{id}, t_1.A_1,...,t_{nc}.A_{nc}\}.$

We conduct a test on the rules of the query depending on our results in Fig. 5. To avoid the loss of data, the operators OPTIONAL and BOUND are used for handling null-values in the SPARQL expressions. In this work, we rewrite the SPARQL queries corresponding to the most important operations of relational algebra: **Selection** (σ), **Projection** (Π), **Rename** (ρ), **Union** ($\cup$), **Difference** ($\setminus$), **Natural Join** (∞), **Left Join** ($\propto$), and **Binary Relation**.

5.1. Rules for Basic Relational Algebra Operations

Rule7 (Selection **(σ)-restriction):** The selection σ is a unary operation in relational algebra. The expression

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

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 (7).

Rule No.	**SQL–Algebra** Q	**SPARQL** Q°
R_7.1	$\sigma_{A_i=v}(TB)$	(GP **FILTER** ($?A_i = v$)) or (GP **FILTER** regex($?A_i, v$)) : $A_i \in att(TB), ?A_i \in \text{var}(GP)$
R_7.2	$\sigma_{A_i=v1 \text{ and } A_k \geq v2 \text{ or } A_d > 01\text{–}01\text{–}2018}(TB)$	(GP **FILTER** ($?A_i =v1$ && $?A_k \geq v2$ || $?A_d$ > "2018-01-01"^^xsd:date))
R_7.3	$\sigma_{IsNull(A_i)}(TB)$	(GP **FILTER** (!bound($?A_i$)))
R_7.4	$\sigma_{IsNotNull(A_i)}(TB)$	(GP **FILTER** (bound($?A_i$)))
R_7.5	$\sigma_{A_i \text{ like } *}(TB)$	(GP **FILTER** regex($?A_i$,""))
R_7.6	$\sigma_{A_i \text{ IN } (v_1,..,v_n)}(TB)$	(GP **FILTER** ($?A_i$ IN(v_1,...,v_n)))

These rules (R_7.1 - R_7.6) cover several cases of SQL condition types, and satisfy the condition (7) using SPARQL FILTER, which restricts the solutions of a query by imposing constraints on values of bound variables. More details are described in the following example.

$$\sigma_{Name="Zhang"}(\text{Professor})$$

This expression returns all rows of the table Professor where the column Name equals “Zhang”. Since an OWL:Class has been created for each table in the RDB, a similar constraint for the objects of this class should be applied to obtain the corresponding result. In order to transform this expression to equivalent SPARQL query satisfying the condition (7), rules R_7.1 is applied. Then the OPTIONAL OPT operator has been added (line 4) to avoid loss of information. It should be observed that the OPT is not added to line 2 or 3, because their predicates (NS:Professor.Prof_No and NS:Professor.Name) are generated from primary key (Prof_No) and Name = “Zhange” constraints respectively. More details are shown in Fig. 6 in equivalent SPARQL query (Q°).

```
SELECT ?Prof_No ?Name ?Research_Direction
WHERE {?S, a , NS:Professor.
                {?S, NS:Professor.Prof_No, ?Prof_No.}
                {?S,NS:Professor.Name,Name.}
OPT{?S,NS:Professor.Direct_Research,?Research_Direction.}
                FILTER(regex(?Name,"Zhang")) }
```

Fig. 6. Q° selection operation algebra $\sigma_{Name="Zhang"}(\text{Professor})$

where a is abbreviate the rdf:type in SPARQL and PREFIX NS:<http://www.mo_exp.edu.cn/ dbLab/#>. As for the query described above, the SPARQL query representing the selection contains three main clauses (SELECT, WHERE and FILTER). In the first line of the WHERE clause, the result set is restricted to contain only object of the type NS:Professor, having an origin in our RDBLAB in the Professor table. This can be useful for accelerating and getting accurate results, because it limits the search scope inside RDF triples, especially when generated from a big RDB. Although our rules in Section 4 do not transform null values, we can still handle null-value in query expressions. The same results are obtained if the rows are returned by DBMS on expression of relational algebra query. Therefore, the results satisfy the definition condition (7), and the query preservation is true.

Rule8 (Projection $\prod$): This rule used to rewrite a SPARQL query (Q°) corresponding to relational algebra (Q) operation (projection $\prod$), which selects of the relevant attributes of a relation. Then the equivalent rule query Q° to satisfy the defining condition (7) can be defined as:

Rule8 (R_8): Projection $\prod$		
	SQL–Algebra Q	**SPARQL** Q°

R_8	$\prod_{TB.A1,\ldots,TB.An}(TB)_{\text{Order by TB.Ai [ASC\|DESC]}}$	**Select** *?TB.A1 ... ?TB.An where {GP}* order by [ASC|DESC](?TB.Ai)
Example	$\prod_{Lab.Lab\_No,Lab.Lab\_Name}(Lab)$	**SELECT** ?Lab_No, ?Lab_Name Where { ?x a NS:Lab. ?x NS:Lab.Lab_No ?Lab_No. OPT{?x NS:Lab.Lab_Name ?Lab_Name.}}

This query involves three triple patterns (TPs) concatenation via AND (.) located at the end of triple pattern. The result set is restricted to objects of the NS:Lab class in the first TP1 {?x a NS:Lab} and assigned to the variable ?x. The variable ?x has the same values that match each triple patterns (TP2 and TP3). The TP2 {?x NS:Lab.Lab_No ?Lab_No.} returns all the mapped values generated from column Lab_No to predicate NS:Lab.Lab_No based on values in subject ?x and binding them in variable ?Lab_No. While, the TP3 OPT{?x NS:Lab.Lab_Name ?Lab_Name.} returns all the values generated from column Lab_Name even the null-value (due to addition of OPT clause) based on values in subject ?x. Since the column Lab_Name may contain null-value, the OPT clause is added to TP3, while the column Lab_No is a PK, so that an OPT is not mandatory to be added. The result returned by SPARQL is the same result returned by DBMS on expression of relational algebra after applying this rule. Therefore, this rule satisfies the defining condition (7).

Rule9 (Rename ρ): In the in relational algebra, a rename ρ operation used to renames one column to another name and projects all columns of Q. Then the equivalent rule query Q° to satisfy the defining condition (7) can be defined as:

Rule9(R_9): Rename ρ	
SQL–Algebra Q	$\prod_{\rho(TB.A1,\ldots,TB.An)\mapsto(B1,\ldots,Bn)}(TB)$
SPARQL Q°	**Select** (*?TB.A1* **AS** *?B1*) ... (*?TB.An* **AS** *?Bn*) *Where*{*GP*}

Obviously, the SPARQL expression (Q°) correspond to relational algebra (Q), that because the rename operator in (Q) and (Q°) will are making the same purpose of the rename operation.

Rule10 (Union $\cup$): A union $\cup$ is one of most common operators, which merges the results returned by two or more projects ($\prod$)-Select statements. Then the equivalent rule query Q° to satisfy the defining condition (7) can be defined as:

Rule10(R_10): Union $\cup$	
SQL–Algebra Q	**SPARQL** Q°
$\prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1)\cup\ldots\cup\prod_{TB_n.A_1,\ldots,TB_n.A_{nc}}(TB_n)$	*Select * where* {{*Select ?TB1.A1 ... ?TB1.Anc where* {*GP*}} **UNION...UNION** {*Select ?TBn.A1 ... ?TBn.Anc where* {*GP*}}}

It can be noticed that the Q expression unifies all rows from the attributes ($A_1,..,A_n$) in TB_1 to TB_n relation. The first requirement is to perform the projection within the UNION clause to restrict the ($?A_1,..,?A_n$) variables for both ($A_1,..,A_n$) attributes. Therefore, the SPARQL query (Q°) will return all values ($A_1,..,A_n$) that originated in TB_1 to TB_n.

Rule11 (Difference \): A difference \ operator is used to minus the result-set of two tables (TB_1 and TB_2). The two tables (TB_1 and TB_2) should have the same columns. The result-set of $TB_1 \setminus TB_2$ is table, which contains all tuples in TB_1 but not in TB_2. Then the equivalent rule query Q° to satisfy the defining condition (7) can be defined as:

Rule11 (R_11): Difference \	
SQL–Algebra Q	**SPARQL** Q°
$\prod_{TB_1.A_1,\ldots,TB_1.A_{nc}}(TB_1)\setminus\prod_{TB_2.A_1,\ldots,TB_2.A_{nc}}(TB_2)$	**Select** ?A1...?An Where {**?x** a NS:TB1. **?x** NS:TB1.A1 ?A1.... **FILTER NOT EXISTS**{*?y* a NS:TB2. *?y* NS:TB2.A1 **?x**.}}

The SPARQL query and its result are represented in the following example (Fig. 7):

Q_1	$\prod_{Lab.Lab\_No}(Lab) \setminus \prod_{Student.Lab\_No}(Student)$	
Q°_1	**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	**Lab_No** "440"^^xsd:string "441"^^xsd:string

Fig. 7. SPARQL query with its results of SQL difference algebra in Q1

So far, in the equivalent SPARQL query, the object NS:Lab and predicate NS:Lab.Lab_No are represented by the variables ?x and ?Lab_No respectively, while the NS:Student object is represented by ?y. The triple pattern {?y NS:Student.Lab_No ?x.} is conditional clause that reflects the relationship between two classes NS:Student and NS:Lab through two variables ?y and ?x, where ?x in TP {?y NS:Student.Lab_No ?x.} refers to ?x in TP {?x a NS:Lab}. That means the foreign key holds and our approach for transformation of relationships (through FK) is satisfied. The FILTER NOT EXISTS expression tests the existence of a pattern {?y NS:Student.Lab_No ?x} in the data, given the bindings already determined by the query pattern {?x a NS:Lab}. This rule uses the object property Student.Lab_No that is generated from *fk(Lab_No,Student)* by Rule 5.2 in Section 4.1, reflecting the integrity of our rules.

5.2. Rules to Generate Q° from a Relational Algebra Join

An SQL join clause is one of the important concepts in relational algebra that is used to combine rows from two tables or more based on a common field between them.

Rule12.1 (Natural join ∞): It is a binary operator, which used to combine tuples from two/more tables $TB_1 \infty TB_2 \infty ... TB_n$, through on a common field between them. The result of $TB_1 \infty TB_2$ is a set of all combination tulpes in TB_1 and TB_2 that are equal on their common attribute names. To reflect the role of foreign keys in the RDB data, SPARQL queries are applied on the resulted ontology to locate tuples (results obtained by variables in SELECT clause). These tuples used to connect each other by object properties (corre sponding to FKs in RDB). Then the equivalent rule query Q° to satisfy the defining condition (7) can be defined as:

Rule12.1 (R_12.1): (Natural join ∞)	
SQL–Algebra Q	**SPARQL** Q°
$\prod_{TB_1.A_1, \mathbf{TB_1.A_i}, ..., TB_1.A_{n1}, TB_2.B_1, \mathbf{TB_2.A_i}, ..., TB_2.B_{n2}}(TB_1 \infty TB_2)$ Where $TB_1.A_i$ and $TB_2.A_i$ are common attributes.	*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 $NS:TB_1.A_i$ $?TB_1.A_1$.*}…**Optional**{*?x $NS:TB_1.A_{n1}$ $?TB_1.A_{n1}$.*} **?x NS:TB_1.A_i ?TB_1.A_i. ?TB_1.A_i** *a $NS:TB_2$.* **Optional**{**?TB_1.A_i** *$NS:TB_2.B_1$ $?TB_2.B_1$.*}…**Optional**{**?TB_1.A_i** *$NS:TB_2.B_{n2}$ $?TB_2.B_{n2}$.*}}

The graph pattern GP {?x NS:TB_1.A_i ?TB_1.A_i. ?TB_1.A_i a NS:TB2.} has an object property (NS:TB_1.A_i) represented by the variable ?TB_1.A_i, which used to link between two classes $NS:TB_1$ and $NS:TB_2$. To ensure no any data lose of triples, the **OPTINAL** operator for all properties of class was used except on the common property that should be its value is not-null. If we change the rule condition by **BOUND** operator in part $(?x\ NS:TB_1.A_i\ ?TB_1.A_i)$ to $\{?x\ a\ NS:TB_1\{\ldots Optional\{?x\ NS:TB_1.A_i\ ?TB_1.A_i.\}\ FILTER(Bound(?TB_1.A_{i2})\}\ldots\}$ the same result was observed. The following Q2 example with it equivalent SPARQL (Q°_2) and its result (Fig. 8) that is generated with input:

Q_2	$\prod_{Student.Stude\_Id, Student.Name, Lab.Lab\_Name}(Student \infty Lab)$

$Q°_2$	**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

Stud_Id	name	lab_no	lab_name
"1"^^xsd:int	"Mohamed"^^xsd:int	NS:Lab_442	"IDC"^^xsd:string
"2"^^xsd:int	"Ali"^^xsd:string	NS:Lab_443	"DCS"^^xsd:string
"3"^^xsd:int	"Shadi"^^xsd:string	NS:Lab_442	"IDC"^^xsd:string
"4"^^xsd:int	"Qing Li"^^xsd:string	NS:Lab_443	"DCS"^^xsd:string
"5"^^xsd:int	"shahd"^^xsd:string	NS:Lab_444	

Fig. 8. SPARQL query results of natural join in Q2

From the above result of SPARQL query, it can be noticed that our rule is satisfied. Moreover, the tuple 5 in Fig. 8 is not lost even the column Lab_Name with null-value due to the use of OPT operator. Therefore, this rule satisfies the defining condition (7).

Rule12.2 (Left Join $\propto$): To satisfy the defining condition (7) according to left join, the equivalent query $Q°$ can be defined as follows:

Rule12.2 (R_12.2): Left Join $\propto$	
SQL–Algebra Q	**SPARQL** $Q°$
$\prod_{TB_1.A_1,\mathbf{TB_1.A_i},\ldots,TB_1.A_{n1},TB_2.B_1,\mathbf{TB_2.A_i},\ldots,TB_2.B_{n2}}(TB_1 \propto TB_2)$ $TB_1.A_i$ and $TB_2.A_i$ are common columns in this expression	*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 $NS:TB_1.A_i$? $TB_1.A_i$.}…**Optional**{?x $NS:TB_1.A_{n1}$ $?TB_1.A_{n1}$.} **Optional**{ **?x NS : TB_1.A_i** **?TB_1.A_i**. **?TB_1.A_i** *a* $NS:TB_2$. **Optional**{**?TB_1.A_i** $NS:TB_2.B_1$ $?TB_2.B_1$.}…} **Optional**{**?TB_1.A_i** $NS:TB_2.B_{n2}$ $?TB_2.B_{n2}$.} }

The different between this rule and Rule 12.1 is the use of nested OPTIONAL. The condition graph pattern GP {?x NS:TB_1.A_i **?TB_1.A_i**. **?TB_1.A_i** a NS:TB_2.} is putted inside OPTIONAL operator in order to avoid losing tuples and guarantee the result of a left outer join for class NS:TB_1 and NS:TB_2. This will further guarantee to obtain all tuples of the "left" class (NS:TB_1), even if the join-condition does not match any tuple in the "right" class (NS:TB_2). For instance, Q3 represents the left join algebra and its corresponding SPARQL query ($Q°_3$). The results of equivalent SPARQL query of SQL left join expression in Q3 is shown in Fig. 9.

Q_3	$\prod_{Student.Stude\_Id,Student.Name,Lab.Lab\_Name}(Student \propto Lab)$
$Q°_3$	**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.} **Optional**{ **?x NS : Student.Lab_No ?lab_no.** **?lab_no** a NS:Lab. **Optional**{ ?lab_no NS:Lab.Lab_Name ?lab_name.}}

Stud_Id	name	lab_no	lab_name
"1"^^xsd:int	"Mohamed"^^xsd:int	NS:Lab_442	"IDC"^^xsd:string
"2"^^xsd:int	"Ali"^^xsd:string	NS:Lab_443	"DCS"^^xsd:string
"3"^^xsd:int	"Shadi"^^xsd:string	NS:Lab_442	"IDC"^^xsd:string
"4"^^xsd:int	"Qing Li"^^xsd:string	NS:Lab_443	"DCS"^^xsd:string
"5"^^xsd:int	"shahd"^^xsd:string	NS:Lab_444	
"6"^^xsd:int	"Abdullah"^^xsd:string		

Fig. 9. SPARQL results of left join algebra in Q3

The condition GP {?x NS:Student.Lab_No **?lab_no. ?lab_no** a NS:Lab.} is included inside OPTIONAL to satisfy our rule and avoid the loss of tuples (e.g. the tuple 5 and 6 in Fig. 9 are not lost). Therefore, this rule satisfies the defining condition (7).

5.3. Rules to Generate $Q°$ from a Binary Relation

Assume that Q is a query algebra over binary relation ***BinaryRel***($TB3, A, B, TB1, A, TB2, B$) and ***ISBinaryRel***($TB3$) holds, based on our rules (R_5.3 and R_6). These rules make the property A be part of the class $TB2$, and property B is part of the class $TB1$, where A and B refers to its class (Fig. 5). The binary

relation is satisfied because it has two FKs, which form PK of the binary relation. The null-value cannot be allowed. Therefore, there is no need to put the triple pattern that is used to connect between the classes (TB_1 and TB_2) through their object properties (A and B) in the **OPTIONAL** operator. Then the equivalent query Q° to satisfy the defining condition (7) can be defined as:

Rule13 (R_13): Rules13 for a Binary Relation	
SQL–Algebra Q	**SPARQL** Q°
BinaryRel $(TB3, A, B, TB1, A, TB2, B)$ *and* ***ISBinaryRel*** $(TB3)$	*Select ?A ?B Where* {**?A** *a* $NS:TB_1$. **?A** $NS:TB_1$.B **?B**. **?B** *a* $NS:TB_2$. **?B** $NS:TB_2.A$ **?A**.}

In this rule, ?A and ?B are variables for representing conditional clauses to produce the structure of the equivalent SPARQL query according to the expression of the binary relation algebra. Furthermore, to obtain the same results retrieved by the expression query of the relational algebra in the RDB. For example, SPARQL query (Q°_4) corresponding algebra (Q4) shows the validity of this rule and the results of Q°_4 are shown in Fig. 10.

Q_4 *SELECT Student Stud _ Id, Student.Name, Courses.`Cors _ No`, Courses.Cors _ Name,*
Student.Name, age FROM (*Stud _ Cors INNER JOIN Student ON Stud _ Cors.Stud _ Id = Student.Stud _ Id*) *INNER JOIN Courses ON Stud _ Cors.Cors _ No = Courses.Cors _ No*

Q°_4

```
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.}
```

Stud_Id	name	Cors_No	Cors_Name
"1"^^xsd:int	"Mohamed"^^xsd:int	"1"^^xsd:int	"IR"^^xsd:string
"2"^^xsd:int	"Ali"^^xsd:string	"1"^^xsd:int	"IR"^^xsd:string

Fig. 10. SPARQL Q°_4 results of binary relation in Q4

Accordingly, it seems fair to conclude that the transformation rules are satisfied the condition definition (7) and validated the rules in Section 4, thus reflecting our integrated rules. Furthermore, the transformation rules (Section 4 and 5) designed in unambiguous structure and kept the tracks of attribute keys in the tables. Therefore, these rules can be extended to produce relational databases from ontologies.

6. Implementation and Comparison

We propose to implement this method in two phases. The first phase is to transform an RDB schema and data to an OWL and RDF triples. Also, we use ontology validator [35] to validate, and show the triple of data model and ontology graph. The second phase is to perform equivalent semantic query on the ontology (RDF triples) generated from the first phase. In the two phases, we use Apache Jena 2.11.0 in Java Language (NetBenas IDE7.3.1). Apache Jena™ is a Java framework that used for building semantic web applications. It provides a set of tools and Java libraries for developing semantic web and linked-data apps, tools and servers [36]. Beside the implementation, experimental analysis is performed to reflect the effectiveness of our work. Moreover, this work also provides a view of comparison with the proposed techniques.

6.1. Implementation

6.1.1. Transforming an RDB to OWL/RDF Triples

For the implementation of this phase, we propose a technique using Apache Jena framework in Java (package-com.hp.hpl.jena). This package consist of interfaces that representing models, resources, properties, literals, and statements. Furthermore, it includes other key concepts of OWL/RDF, and a ModelFactory for models creation. To examine our implementation, the new method should be applied to

an RDB. Fig. 2 shows a sample RDB named, RDBLAB (have several cases of RDB concepts), created by MYSQL5.6 and connected with Java JDBC by "com.mysql.jdbc.Driver". For syntactic ontology validation, RDF validator is a tool that used to test the RDF/XML syntax documents (passed as parameters), displays a tabular presentation and graphical of this documents. The RDF validator is used to validate the generated results. As a result of this phase, Fig. 5 (RDB data and RDF data model) illustrates the use of transformation rules to export an RDB data from a database (RDBLAB) into ontology as RDF triples. The left side of Fig. 5 shows an RDB data of relationships (1:1, 1:M and N:M) between five tables (Student, Stud_Cors, Courses,Lab and Professor) and the right side shows corresponding OWL/RDF ontology. In addition, to understand the ontology (RDF(S)-OWL) formation, the ontology code production and the ontology graph representation, we used RDF validator to validate RDF(S)-OWL code that generated after applying our rules on RDB, and showed the RDF triples and graph as resulting ontology. According to the schema and data of the RDBLAB (Figs. 2 and 5), the results are shown in Figs. 5 (right side), 11 and 12.

6.1.2. Semantic query on the OWL/RDF triples

In phase two, we propose a method implemented using Apache Jena 2.11.0 package ARQ API [36], [37] (a SPARQL 1.1). ARQ is a query engine for Jena, which supports the SPARQL RDF query language. In the ARQ API, the key semantic query (SPARQL) package for the application developer is jena-arq-2.11.0.jar (com.hp.hpl.jena.query). This package contains interfaces for representing Query, QueryExecution, QueryExecution Factory, QueryFactor, ResultSetFormatter, and all the other key concepts of SPARQL. Fig. 13 shows our interface of semantic query on an RDF triples file that is generated from the phase one. Our implemented tool also contains a direct SPARQL query endpoint so that the user can directly run the SPARQL queries. All query results that are shown in Section 5 proved the efficiency of our approach, and verified the preservation of semantics with SPARQL.

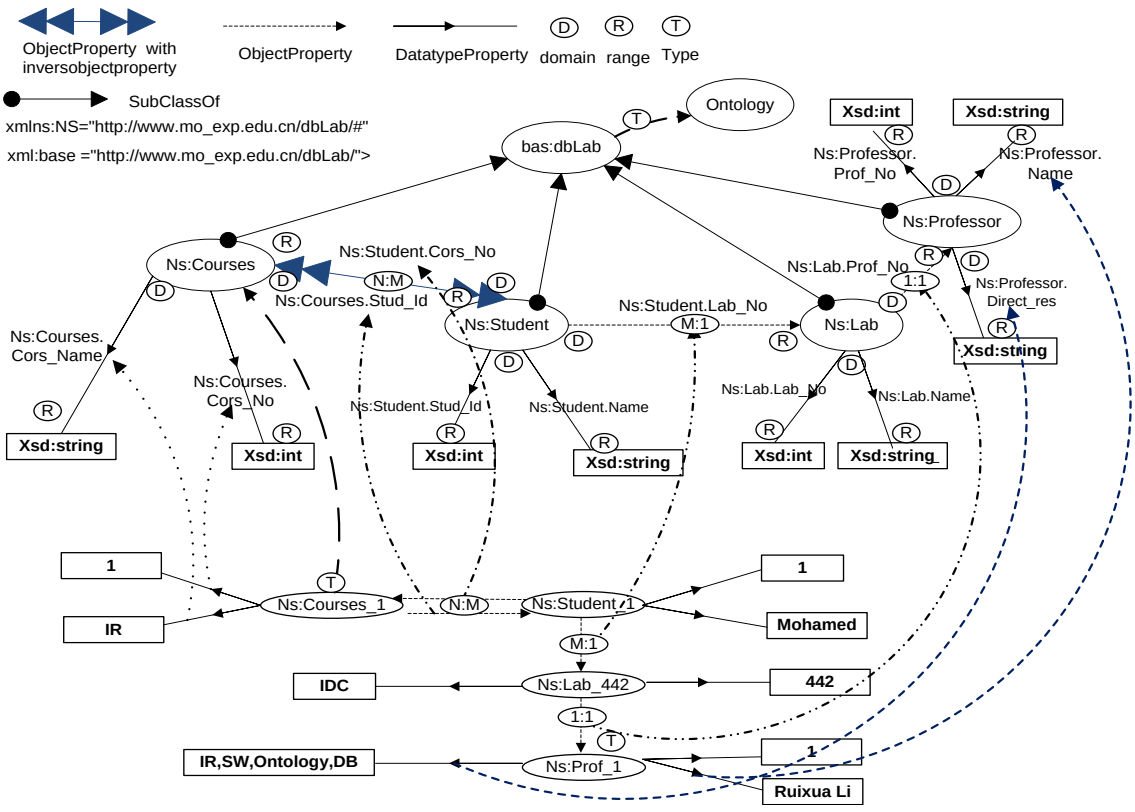


Fig. 11. Graph of ontology extracted from RDLAB (Figs. 2 and 5) schema and data instance

Ontology Schema

```
<?xml version="1.0"?>
...
xmlns="http://www.mo_exp.edu.cn/dbLab/#"
xmlns:NS="http://www.mo_exp.edu.cn/dbLab/#"
xml:base ="http://www.mo_exp.edu.cn/dbLab/">
<owl:Ontology rdf:about="dbLab"/>
<owl:Class rdf:ID="Student"> <rdfs:subClassOf rdf:resource="dbLab"/> </
owl:Class>
<owl:Class rdf:ID="Courses">...</owl:Class>
<owl:Class rdf:ID="Lab">...</owl:Class>
<owl:Class rdf:ID="Professor">...</owl:Class>
<owl:Class rdf:ID="Postion">...</owl:Class>
<!-- Student ontology Schema RDFS-OWL -->
<owl:InverseFunctionalProperty rdf:ID="Student.Stud_Id"/>
<owl:datatypeproperty rdf:ID="Student.Name">
  <rdfs:label> Student Name</rdfs:label>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdfs:comment>Used to store the Name of Student</rdfs:comment>
</owl:datatypeproperty>
<owl:ObjectProperty rdf:ID="Student.Cors_No">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Courses"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Student.Lab_No">
  ...
<owl:ObjectProperty rdf:ID="Student.Post_No">
  ...
<owl:Class rdf:about="#Student">
...
    <owl:Restriction>
      <owl:onProperty rdf:resource="#Student.Stud_Id"/>
      <owl:minCardinality rdf:datatype="&xsd;int">1</owl:minCardinality>
    ...
      <owl:onProperty rdf:resource="#Student.Cors_No"/>
      <owl:allValueFrom  rdf:resource="#Courses"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</...>
    ...
      <owl:onProperty rdf:resource="#Student.Lab_No"/>
      <owl:allValueFrom  rdf:resource="#Lab"/>
      ...
      <owl:onProperty rdf:resource="#Student.Post_No"/>
      <owl:hasValue  rdf:resource="#Postion"/>
      ...
<!--Schema for Courses -->
<owl:InverseFunctionalProperty rdf:ID="Courses.Cors_No"/>
<owl:datatypeproperty rdf:ID="Courses.Cors_Name">
  ...
<owl:ObjectProperty rdf:ID="Courses.Stud_Id">
  <rdfs:domain rdf:resource="#Courses"/>
  <rdfs:range rdf:resource="#Student"/>
  ...
      <owl:onProperty rdf:resource="#Courses.Stud_Id"/>
      <owl:allValueFrom  rdf:resource="#Student"/>
      ...
<!--Schema for Lab -->
<owl:InverseFunctionalProperty rdf:ID="Lab.Lab_No"/>
...
<owl:ObjectProperty rdf:ID="Lab.Prof_No">
  <rdfs:domain rdf:resource="#Lab"/>
  <rdfs:range rdf:resource="#Professor"/>
...
      <owl:onProperty rdf:resource="#Lab.Prof_No"/>
      <owl:hasValue  rdf:resource="#Professor"/>
    ...
</rdf:RDF>
```

Ontology instance(RDF Triples)

```
<!-- Student Instances -RDF TRIPLES -->
<Student rdf:ID="Student_1">
<NS:Student.Stud_Id rdf:datatype="&xsd;int">1</NS:Student.Stud_Id>
<NS:Student.Name rdf:datatype="&xsd;string">Mohamed</NS:Student.Name>
<NS:Student.Lab_No rdf:resource="#Lab_442"/>
<NS:Student.Cors_No rdf:resource="#Cors_1"/>
</Student>
...
<Student rdf:ID="Student_6">
<NS:Student.Stud_Id rdf:datatype="&xsd;int">6</NS:Student.Stud_Id>
<NS:Student.Name rdf:datatype="&xsd;string">Abdullah</NS:Student.Name>
</Student>
...
<!--Courses Instance-->
<Courses rdf:ID="Cors_1">
<NS:Courses.Cors_No rdf:datatype="&xsd;int">1</NS:Courses.Cors_No>
<NS:Courses.Cors_Name rdf:datatype="&xsd;string">IR</
NS:Courses.Cors_Name>
<NS:Courses.Stud_Id rdf:resource="#Student_1"/>
<NS:Courses.Stud_Id rdf:resource="#Student_2"/>
</Courses>
<Courses rdf:ID="Cors_2">
<NS:Courses.Cors_No rdf:datatype="&xsd;int">1</NS:Courses.Cors_No>
<NS:Courses.Cors_Name rdf:datatype="&xsd;string">DBMS</
NS:Courses.Cors_Name>
</Courses>
...
<!--Lab Instance-->
<Lab rdf:ID="Lab_440">
<NS:Lab.Lab_No rdf:datatype="&xsd;int">440</NS:Lab.Lab_No>
<NS:Lab.Lab_Name rdf:datatype="&xsd;string">SWE</NS:Lab.Lab_Name>
<NS:Lab.Prof_No rdf:resource="#Professor_5"/>
</Lab>
...
<Lab rdf:ID="Lab_442">
<NS:Lab.Lab_No rdf:datatype="&xsd;int">442</NS:Lab.Lab_No>
<NS:Lab.Lab_Name rdf:datatype="&xsd;string">IDC</NS:Lab.Lab_Name>
<NS:Lab.Prof_No rdf:resource="#Professor_1"/>
</Lab>
...
<Lab rdf:ID="Lab_444">
<NS:Lab.Lab_No rdf:datatype="&xsd;int">444</NS:Lab.Lab_No>
<NS:Lab.Prof_No rdf:resource="#Professor_4"/>
</Lab>
<!--Professor Instance-->
<Professor rdf:ID="Professor_1">
<NS:Professor.Prof_No rdf:datatype="&xsd;int">1</NS:Professor.Prof_No>
<NS:Professor.Name rdf:datatype="&xsd;string">Ruixuan Li</
NS:Professor.Name>
<NS:Professor.Direct_Research
rdf:datatype="&xsd;string">IR,SW,Ontology,DB</
NS:Professor.Direct_Research>
</Professor>
...
<Professor rdf:ID="Professor_5">
<NS:Professor.Prof_No rdf:datatype="&xsd;int">5</NS:Professor.Prof_No>
<NS:Professor.Name rdf:datatype="&xsd;string">Jing</NS:Professor.Name>
<NS:Professor.Direct_Research rdf:datatype="&xsd;string">MDBS</
NS:Professor.Direct_Research>
</Professor>
```

Fig. 12. Part of ontology corresponding to the (Figs. 2 and 5) RDB schema and data of relationships (1:1, 1:M and N:M).

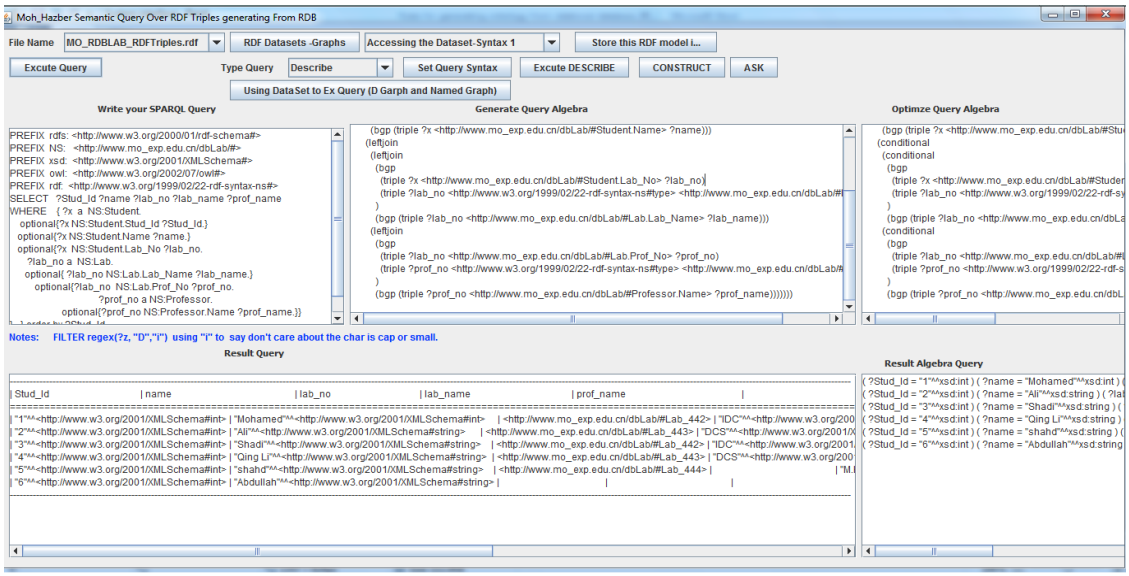


Fig. 13. Semantic query interface over RDF triples

Finally, this section provides some examples of SPARQL queries to interrogate the resulting ontology by analogy to the interrogation of the RDB by the SQL queries. For example, the SPARQL query (Q°_5) corresponding to the query in relational algebra (Q_5). Query(Q_5) 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. 14 shows the results returned by the execution Q5 using RDBMS (MySQL) and Fig. 15 shows the corresponding results returned by the execution equivalent SPARQL query ($Q°_5$) using our system interface (Fig. 13), which used as the interfaces to execute a SPARQL query on RDF triples for querying generated ontologies. From Fig. 14 and 15 it can be observed that the returned dataset of Q5 and its equivalent ($Q°_5$) are same and this reflects the validity of our approach.

Q_5 $\prod_{Stud\_Id,Lab.Lab\_No,Prof.Name}((Student \propto Lab) \propto Professor)$

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

Fig. 14. LEFT JOIN (Q5) and the result returned by DBMS

$Q°_5$

```
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
```

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			

Fig. 15. SPARQL $Q°_5$ result corresponding SQL algebra (Q5)

To more reflect the validity of our approach, an extra example (Q6) has added and its result is shown in Fig. 16(a)(b) by using different rules (R_8, R_9, and R_10). This query (Q6) used three rules for rewriting SPAQRL($Q°_6$) query corresponding to the operations of SQL algebra (projection, rename, and union).

Q_6 $\prod_{\rho(Stud\_Id,Name,'Stud')\mapsto(id,name,type)}(Student) \cup \prod_{\rho(Lab\_No,Lab\_Name,'Lab',)\mapsto(id,name,type)}(Lab) \cup \prod_{\rho(Prof\_No,Name,'Prof')\mapsto(id,name,type)}(Professor)$

$Q°_6$

```
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
```

id	name	type
1	Mohamed	Stud
1	Ruixuan Li	Prof
2	Ali	Stud
2	Zhang	Prof
3	Lu yuan	Prof
3	Shadi	Stud
4	M.Hazber	Prof
4	Qing Li	Stud
5	Shahd	Stud
5	Jing	Prof
6	Abdullah	Stud
440	SWE	Lab
441	AI	Lab
442	IDC	Lab
443	DCS	Lab
444	NULL	Lab

(a) Result from RDB data using SQL

id	name	type
"1"^^xsd:int	"Mohamed"^^xsd:string	"Stud"
"1"^^xsd:int	"Ruixuan Li"^^xsd:string	"Prof"
"2"^^xsd:int	"Ali"^^xsd:string	"Stud"
"2"^^xsd:int	"Zhang"^^xsd:string	"Prof"
"3"^^xsd:int	"Lu yuan"^^xsd:string	"Prof"
"3"^^xsd:int	"Shadi"^^xsd:string	"Stud"
"4"^^xsd:int	"M.Hazber"^^xsd:string	"Prof"
"4"^^xsd:int	"Qing Li"^^xsd:string	"Stud"
"5"^^xsd:int	"Jing"^^xsd:string	"Prof"
"5"^^xsd:int	"shahd"^^xsd:string	"Stud"
"6"^^xsd:int	"Abdullah"^^xsd:string	"Stud"
"440"^^xsd:int	"SWE"^^xsd:string	"Lab"
"441"^^xsd:int	"AI"^^xsd:string	"Lab"
"442"^^xsd:int	"IDC"^^xsd:string	"Lab"
"443"^^xsd:int	"DCS"^^xsd:string	"Lab"
"444"^^xsd:int		"Lab"

(b) Result from ontology data using SPARQL

Fig. 16. SPARQL($Q°_6$) result corresponding SQL algebra (Q6)

From the previous results, it can be seen obviously that the ontology results (Figs. 5, 11, and 12) and queries (Q1-Q6) are well integrated. During the transformation process, there is no data loses, even for the null values. Moreover, the combination of ontology (schema and instance) and SPARQL query have the ability to provide the same results of RDB (schema and data) using SQL query algebra.

6.2. 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. For validating the efficiency of this work in terms of quantitative, Table 1 shows the dataset of RDBLAB row tables (100200) including null-values in each column tables. Furthermore, it presents quantity of rows in which null-values that appears through the relationship between the tables. Additionally, the null-values in this table reflect

the size of data that are not lost when our rules are applied. Moreover, the table presents the corresponding number of tuple classes returned from RDF triples (752916) that are generated from dataset of RDBLAB. We also emphasize the validity and accuracy of our rules by applying all the queries (Q1-Q6) on the new dataset. To present the significance of our work, additional SPARQL (Q`) queries are used for reflecting the volume of data loss compared with our approach SPARQL (Q°). The SPARQL (Q°) are modified from the original SPARQL (Q°) queries (Table 2). The SQL(Q1-Q6) queries are executed on the RDBLAB dataset using RDBMS (MYSQL 5.6), while the SPARQL (Q° and Q`) queries are executed on RDF triples (generated from RDBLAB dataset). The results of queries and their execution time are shown in Table 3. Table 3 shows quantitative analyses of the dataset, which are represented in the Figs. 17 and 18. It can be seen obviously from Fig. 17 that the results obtained from SQL(Q) and SPARQL(Q°) are the same results, and there are no data losses from SQL(Q) compared to SPARQL(Q`) (Figs. 18). Moreover, the performance analysis of different SQL(Q), SPARQL(Q°) and SPARQL(Q`) rules is shown in Fig. 19. It can be observed that, the minimum retuned execution time is obtained by SQL(Q), because it has been carried out by RDBMS (MYSQL 5.6 engine). There is a small increase in returned execution time that is obtained by our rules SPARQL(Q°). Moreover, when the execution time of SPARQL(Q`) compared with our SPARQL(Q°) there is small differences. This because our approach searching all tuples even for that contains null-values, while the SPARQL(Q`) avoid the searching of tuples that contains null-values. All these results together reflect the high accuracy and significance of our work.

Table 1. Table Rows of RDBLAB (Contain Null-value) and Class Tuples of Ontology Generated

Tables	Rows	Columns have null-value	Rows have null-value	SPARQL TP used to return class tuples	Returned tuples
Student	70000	Lab_No=7614 Age=2500	Students with lab_no is null=7614. Students with age is null=2500. Lab_no or age is null =10114 Students with lab_no is not null and lab.lab_name is null=7873. Students with lab_no is not null and lab.prof_no is null=4212. Students with lab_no is not null and lab.lab_name and lab.prof_no is null = 169.	{?x a NS:Student}	70000
Lab	3000	Lab_Name=199 Prof_No=111	Rows of lab used by students =2625. Rows of lab used by students with lab_name is null=190. Lab with lab_name and prof_no is null=25.	{?x a NS:Lab}	3000
Courses	100			{?x a NS:Courses}	100
Stud_Cors	27000			Q4	27000
Professor	100			{?x a NS:Professor}	100

Table 2. 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 3. Results of Queries and their Executing Time

Q	Q Rows	Time (s)	Q° Tuples	Time (s)	Q` Tuples	Time (s)
Q1	375	0.007	375	0.008	366	0.009
Q2	62386	0.046	62386	0.06	54513	0.095
Q3	70000	0.071	70000	0.075	62386	0.1
Q4	27000	0.025	27000	0.035	24190	0.04
Q5	70000	0.07	70000	0.075	58174	0.095

Q6	73100	0.146	73100	0.156	72901	0.157

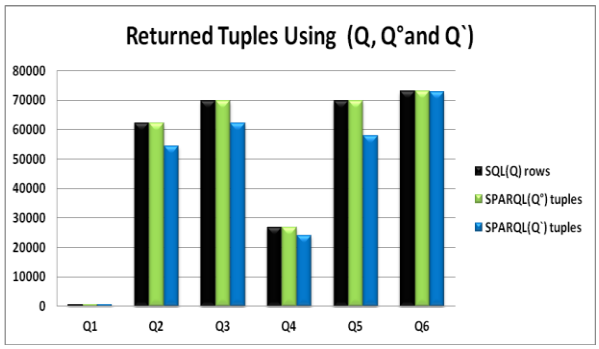


Fig. 17. Comparing between SQL(Q), SPARQL(Q°) and SPARQL(Q`) results

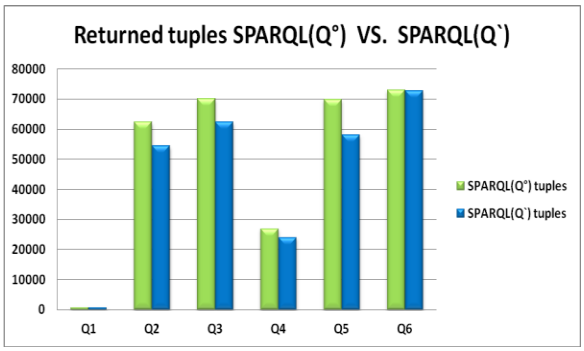
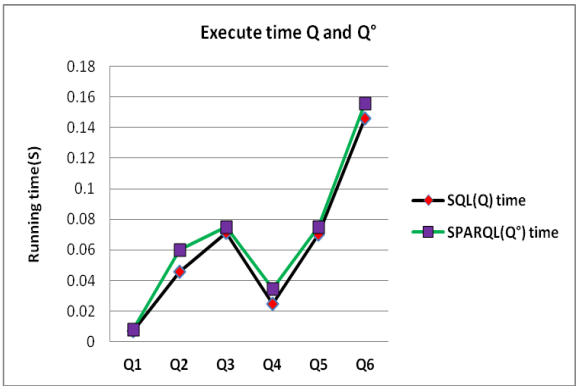


Fig. 18. Comparing between SPARQL (Q°) and SPARQL (Q`) results



(a) SQL(Q) and SPARQL(Q°)

(b) SQL(Q), SPARQL(Q°) and SPARQL(Q`)

Fig. 19. Execute time of Query Statements

6.3. Comparison

Our approach is more advanced, integrated and characterized by different features when compared with the existing approaches (Table 4). These features can be noted as follows.

- At first, we analyze the concepts of RDB and ontology then our analysis indicates the similarities between these concepts, which can be used as the basic in this field for new researches. The analysis used in this approach is more modified when compared with other studies [18], [20]. These studies depend on their analysis upon the properties of RDB and ontology in general but they do not concentrate on the similarities between RDB and ontology.
- Every transformation stage depends on the previous one to improve its integrity and validity in application. Moreover, we use formal rules for all steps including ontology construction, RDF generation and semantic query on RDF triples with illustrated examples and results while the others [11], [19], [38] used informal rules.
- Our work deals with the most important concepts of RDB to generate ontology and its query in the all stages of transformation (schema, data and query) such as primary key, foreign key, constraints and relationships, unlike the others [11], [16], [17], [38] they are not integrated in their approaches (Table 4).
- We consider the data null value transformation and its query. This feature is not studied by formal researches (Table 4).
- Generating IRI for the triples shows the capability of our approach to avoid confusion, provide clear triples of ontology schema, indicate the source of relational schema and also not be allowed to duplicate the names of properties even two or more tables have the same name of a column.

- Our approach for semantic query is based on set of formal rules to rewrite the SPARQL query corresponding SQL relational algebra, with illustrated examples and results that proved the validity of our rules and the possibility of use in real applications. While authors [29], [38] use one example that shows SPARQL query corresponding SQL.
- Our paper deals with the most important concepts of SQL relational algebra operations to generate formal rules of semantic query (SPARQL) on RDF data such as selection, projection, rename, union, difference, natural join, left Join, and binary relation. While authors [29], [38] show the SPARQL syntax query for representing relationship 1:M by one example written by SPARQL and SQL.
- Additional features are shown clearly in Table 4.

Accordingly, our approach shows integrated rules with represented examples, validation, results and implementation. All these indicate the efficiency of our approach and provide the developer of semantic web a clear procedure to develop their applications that depend on RDB.

Table 4. A Comparison between our Proposed Approach and other Existing Approaches

Method	Model	Analysis concepts of RDB and Ont.	Rel. Schema	Rel. Data	Data Null value	Data Transform	Generate Identifier RwId	Formal/ Informal	Validation	Semantic query	Implementation
Buccella et al.	Semi-auto	No	M:N	No	No	No	No	Expository example	No	No	No
Stojanovic et al.	Semi auto	Yes	1:1,1:M	1:M	No	Yes	No	Formal	No	No	No
Astrova et al.	Auto	No	1:1	No	No	Yes(weak)	No	Informal	No	No	Yes
Shen et al.	Semi-auto	Yes	1:M	1:M	No	Yes	No	Informal	No	No	No
Li et al.	Auto	Ontology	M:N	No	No	Yes	No	Informal	No	No	Ontology learning Framework
Mohamed et al.	Auto	No	M:N	No	No	Yes(weak)	No	Informal	No	No	No
Zhang & Li	Semi-auto	Yes(Weak)	1:1	No	No	Yes(weak)	No	Informal	No	No	Yes
Bakkas & Bahaj	Auto	No	M:N	No	Yes	Yes	No	Informal	Yes	simple example	Yes(weak)
Proposed approach	Auto	Yes	1:1,1:M, M:N	1:1, 1:M, M:N	Yes	Yes	Yes	Formal	Yes	Yes	Yes

7. Conclusions

Study on ontology construction of information resources such as RDB is becoming far more widespread in the computer science community. It's helping for the integrating relational databases with Semantic Web and accessing RDB through assisting in the semantic query formulation process. This paper presented a novel method for rewriting equivalent SPARQL query from the SQL - relational algebra query based on the direct mapping rules. Firstly, this paper presented the well-formulated rules for translating RDB schema to OWL ontology and converting RDB instances (considering the null-values) to ontology triples. Secondly, our work proposed a set of rules that transformed relational algebra operation queries into equivalent SPARQL queries. These rules enable executing SPARQL queries over RDB as RDF triples with getting the accurate results without data loss and showed how to deal with null values in the queries according to the structure of SQL query. The effectiveness of the proposed approach is demonstrated with examples and experimental analysis. The integrated ontology and the query results are shown and analyzed. All those indicate the effectiveness and efficiency of the proposed approach. For the limitation of our approach, we did not address on how to deal with queries that contain GROUP BY, HAVING, and SQL aggregation functions, such as SUM(), COUNT(), MIN() and MAX(). In addition, our approach does not explain in details on how to rewrite the equivalent SPARQLs from the nested SQL queries. All these limitations will be considered in the future works.

References

[1] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american, 284*(5), 28-37.

[2] Vasilecas, O., Bugaite, D., & Trinkunas, J. (2006). On Approach for Enterprise Ontology Transformation into Conceptual Model. *Proceedings of the International Conference on Computer Systems and Technologies*(pp. IIIA.23-1- IIIA.23-6). University of Veliko Tarnovo, Bulgaria.

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

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

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

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

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

[8] Geller, J., Chun, S. A., & An, Y. J. (2008). Toward the semantic deep web. *IEEE Computer, 41*(*9*), 95-97.

[9] He, B., Patel, M., Zhang, Z., & Chang, K. C.-C. (2007). Accessing the deep web. *Communications of the ACM, 50*(*5*), 94-101.

[10] Hazber, M. A., Li, R., Gu, X., & Xu, G. (2016). Integration mapping rules: Transforming relational database to semantic web ontology. *Appl. Math, 10*(3), 1-21.

[11] Mohamed, H., Jincai, Y., & Qian, J. (2010). Towards integration rules of mapping from relational databases to semantic web ontology. *Proceedings of 2010 International Conference on Web Information Systems and Mining (WISM)*(pp. 335-339). Sanya, China: IEEE.

[12] Marx, E., Salas, P., Breitman, K., Viterbo, J., & Casanova, M. A. (2013). RDB2RDF: A relational to RDF plug-in for Eclipse. *Software: Practice and Experience, 43*(4), 435–447.

[13] Hert, M., Reif, G., & Gall, H. C. (2011). A comparison of RDB-to-RDF mapping languages. *Proceedings of the 7th International Conference on Semantic Systems*(pp. 25-32). Graz, Austria: ACM.

[14] Stojanovic, L., Stojanovic, N., & Volz, R. (2002). Migrating data-intensive web sites into the semantic web. *Proceedings of the 2002 ACM symposium on Applied computing*(pp. 1100-1107). Madrid, Spain: ACM.

[15] Astrova, I. (2004). Reverse engineering of relational databases to ontologies. In *Proceeding of 1st European Semantic Web Symposium(ESWS)*(pp. 327-341). Heraklion, Grete, Greece: LNCS.

[16] Buccella, A., Penabad, M. R., Rodriguez, F., Farina, A., & Cechich, A. (2004). From relational databases to OWL ontologies. *Proceedings of the 6th National Russian Research Conference on Digital Libraries*. Pushchino, Russia.

[17] Li, M., Du, X.-Y., & Wang, S. (2005). Learning ontology from relational database. *Proceedings of 2005 International Conference on Machine Learning and Cybernetics* (pp. 3410-3415). Guangzhou, China: IEEE.

[18] Shen, G., Huang, Z., Zhu, X., & Zhao, X. (2006). Research on the rules of mapping from relational model to OWL. *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions*(pp. 548-552). Athens, Georgia-USA: CEUR-WS.org *(http://ceur-ws.org/Vol-216/)*.

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

[20] Zhang, L., & LI, J. (2011). Automatic generation of ontology based on database. *Journal of Computational Information Systems, 7*(*4*), 1148-1154.

[21] Hu, W., & Qu, Y. (2007). Discovering simple mappings between relational database schemas and ontologies. *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*(pp. 225-238). Busan, Korea: Springer-Verlag.

[22] Bizer, C., & Seaborne, A. (2004, November). D2RQ-treating non-RDF databases as virtual RDF graphs. *Proceedings of the 3rd International Semantic Web Conference (ISWC2004 posters)* (Vol. 2004). Hiroshima, Japan.

[23] Xu, Z., Zhang, S., & Dong, Y. (2006). Mapping between relational database schema and OWL ontology for deep annotation. *Proceedings of the 2006 IEEE/WIC/ACM international Conference on Web intelligence* (pp. 548-552). Hong Kong: IEEE Computer Society.

[24] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., & Aumueller, D. (2009). Triplify light-weight linked data publication from relational databases. *Proceedings of the 18th International Conference on World Wide Web* (pp. 621–630). Madrid, Spain.

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

[26] Hazber, M. A., Li, R., Xu, G., & Alalayah, K. M. (2016). An Approach for Automatically Generating R2RML-Based Direct Mapping from Relational Databases. *Proceedings of the International Conference of Young Computer Scientists, Engineers and Educator s* (pp. 151-169). Springer.

[27] Hazber, M. A. G., Li, R., Zhang, Y., & Xu, G. (2015). An Approach for Mapping Relational Database into Ontology. *Proceedings of the 12th Web Information System and Application Conference (WISA2015)* (pp. 120-125). Jinan, Shangdong, China: IEEE.

[28] Bizer, C., & Cyganiak, R. (2006). D2r server-publishing relational databases on the semantic web. *Proceedings of the 5th International Semantic Web Conference (ISWC2006)*. Athens, GA, USA.

[29] Banu, A., Fatima, S. S., & Rahman Khan, K. U. (2011). Semantic-based querying using ontology in relational database of library management system. *International Journal of Web & Semantic Technology, 2*(4), 21-32.

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

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

[32] Rodriguez-Muro, M., Rezk, M., Hardi, J., Slusnys, M., Bagosi, T., & Calvanese, D. (2013). Evaluating SPARQL-to-SQL translation in ontop. *Proceedings of the. 2nd OWL Reasoner Evaluation Workshop (ORE 2013)* (Vol. 1015, pp. 94-100). Ulm, Germany: CEUR-WS.org (*http://ceur-ws.org/Vol-1015/ore2013_proceedings.pdf*).

[33] Cyganiak, R. (2005). A relational algebra for SPARQL. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*.

[34] Biron, P. V., & Malhotra, A. (2004). XML schema part 2: Datatypes second edition. from http://www.w3.org/TR/xmlschema-2/

[35] Prud'hommeaux, E., & Lee, R. (2004). W3C RDF validation service. from http://www.w3.org/RDF/Validator/

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

[37] McCarthy, P. (2005). Search RDF data with SPARQL. from http://www.ibm.com/developerworks/library/j-sparql/

[38] BAKKAS, J., & BAHAJ, M. (2013). Generating of RDF graph from a relational database using Jena API. *International Journal of Engineering & Technology (0975-4024), 5*(2), 1970-1975.

Mohamed Hazber is an assistant professor at School of Computer Science, Wuhan University, China. He received the B.S. degree from School of Computer Science and information systems at University of Technology-Baghdad-Iraq 2000 and his M.S. degree from School of Computer Science and Technology at Central China Normal University 2011. He gained PhD degree in School of Computer Science and Technology, Huazhong University of Science and Technology, China in 2016. His research interests include knowledge extraction, relational database, semantic web and ontologies, database and ontology integration.

Bing Li is a professor in School of Computer Science at Wuhan University, Wuhan. He received his Ph.D., M.S., and B.S. degrees, all in computer science, from Huazhong University of Science and Technology, Wuhan, in 2003, 1997, and 1990, respectively. His main research interests include software engineering, service computing, complex system, semantic Web service and artificial intelligence.

Guandong Xu is an associate professor in the School of Software and Advanced Analytics Institute at University of Technology Sydney. He received MSc and BSc degree in Computer Science and Engineering from Zhejiang University, China. He gained PhD degree in Computer Science from Victoria University in 2008. After that he worked as a postdoctoral research fellow in Centre for Applied Informatics at Victoria University and then postdoc in Department of Computer Science at Aalborg University, Denmark. His research interests include web information retrieval, web mining, web services etc.

Mohammed A.S. Mosleh received the B.S. degree from Department of Information Systems at Saba University, Yemen 2009 and his M.S. degree from School of information Technology, Bharathiar University, India in 2012. He received his PhD in computer science from Bharathiar University, India 2017. His research interests include cloud computing, databases.

Xiwu Gu received the BS, MS, and PhD degrees in computer science from Huazhong University of Science and Technology in 1989, 1998, and 2007 respectively. He is currently a lecturer in the School of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China. His research interests include distributed computing, data mining, and social computing. He is a member of the China Computer Federation (CCF).

Yuhua Li is an associate professor in the School of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China. She received the PhD degree in computer science from Huazhong University of Science and Technology in 2006. Her research interests include link mining, social network mining, graph mining, knowledge engineering, Semantic Web and ontology. She is a senior member of the China Computer Federation (CCF).