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

1School of Computer Science, Wuhan University, Wuhan, China
2Advanced Analytics Institute, University of Technology Sydney, Australia
3School of IT & Science, Dr. G.R.Damodaran College of Science, Coimbatore, India
4School 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 underling 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 \( TB(A_1),...TB_n(A_n) \), where \( TB \) is the name of the \( i \)-th relation (table) and \( A \) is the set of its attributes (columns) name that denoted by \( \text{attr}(TB) = \{A_1,...,A_n\} \). and (ii) a relational database RDB over RDBS is a set of instance (data) of relations \( I_1,...,I_m \) over \( TB(A_1),...,TB_n(A_n) \) respectively. Where \( I_1 \) is a set of tuples (rows) \( r_{11},...,r_{1n} \) over \( TB_1 \) that denoted by \( I(TB_1) = \{r_{11},...,r_{1n}\} \in \text{RDB} \in \text{RDBS} \) for all attributes in \( TB_1 \), where each row \( r_{j1},...,r_{jm} \) and \( I(TB_i(A)) = \{r_{j1},...,r_{jm},A\} \in \text{RDB} \in \text{RDBS} \) for \( A \in \text{TB}_i \). The notation \( r_{ij},A \) (or \( \text{val}(A,r_{ij},TB_i) \)) refers to the value of a row \( r_{ij} \) in a column \( A \), (iii) each column \( A \) has a data type \( A_{type} \) i.e. string, int, float, date, etc.). In the following, we underline the attributes of a relation that belong to its primary key \( pk_i(TB_i) \) and we denote for foreign key by \( TB.A \rightarrow B \) (or \( f_k(A,TB_i,B,TB_j) \)) between the attribute \( A \) of a relation \( TB_i \) and the attribute \( B \) of a relation \( TB_j \).

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:

<table>
<thead>
<tr>
<th>Definition 1: SQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select ( A_1,...,A_n ) From ( TB_1,...,TB_n ) Where ( P )</td>
</tr>
</tbody>
</table>

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

**Definition 2:** SQL algebra \( (Q) \)

\[
\prod_{A_1, \ldots, A_m \in \text{att}(T_B \times \ldots \times T_{B_n})} n, m \geq 0
\]

\( \sigma \) stand for a condition like \( \sigma_{A_{i=1}} \): \( x \) is variable or value.

\( A_1, \ldots, A_n \in \text{att}(T_B \times \ldots \times T_{B_n}) \subseteq \text{att}(RDB(T_B \ldots T_{B_n})) \in \text{RDBS} \)

### 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) \( CLS(DTP_i), \ldots, CLS(DTP_m) \), where \( CLS \) 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 \( \text{att}(CLS_i) = \{DTP_i \ldots DTP_m\} \in CLSi \) and (ii) a semantic web ontology \( SWO \) over \( SWOS \) is a set of instance of class \( Io_1, \ldots, Io_n \) over \( CLS(DTP_i), \ldots, CLS(DTP_m) \), respectively. Where \( Io_i \) is a set of tuples (graph) \( tr_{i1}, \ldots, tr_{ir} \) over \( CLS_i \), that denoted by \( Io(CLS_i) = \{tr_{i1}, \ldots, tr_{ir}\} \in SWO \in SWOS \) for all datatype properties in \( CLS_i \), where each triple \( tr_i : l \leq i \leq m \), and \( Io_i(CLSP_i) = \{tr_{i1}, \ldots, tr_{ir}, DTP_i\} \in SWO \in SWOS \) for \( DTP_i \) in \( CLS_i \). The notation \( tr_i.\text{DTP} \) (or \( \text{val}(DTP, tr_i, CLS_i) \)) refers to the value of a tuple (triple) \( tr_i \) in an property (attribute) \( DTP_i \), (iii) each (owl:ObjectProperty) \( OBP \) and \( DTP \) has a set of domain (rdfs:domain) \( \text{Dom} \) and range (rdfs:range) \( \text{Rng} \) classes, and \( DTP \) has an XML schema data type \( \text{DTP}_{\{	ext{xsd}\}} \) [34] (e.g., \( \text{rdf}\text{:resource=}{\text{xsd}\text{:string}}", {\text{xsd}\text{:int}}", {\text{xsd}\text{:float}}", {\text{xsd}\text{:date}}", \ldots\). Each class has a set of object properties \( OBP \) to describe the relations between properties of classes \( CLS(DTP) \) denoted by \( CLS_{\langle \text{dom}\rangle}(DTP) \rightarrow^{\langle \text{dom}\rangle} CLS_{\langle \text{obj}\rangle}(DTP) \) or \( OBP(DTP, CLS_{\langle \text{dom}\rangle}(DTP), CLS_{\langle \text{obj}\rangle}(DTP)) \) that mean \( OBP \) is a relationship between \( CLS_{\langle \text{dom}\rangle}(DTP) \) and \( CLS_{\langle \text{obj}\rangle}(DTP) \) through its domain \( \text{Dom} \) and range \( \text{Rng} \).

**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 \( G = \{t_1, \ldots, t_n\} \). 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 \( ?V_1, \ldots, ?V_n \).

**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 \( \text{SELECT, FILTER, OPT(OPTIONAL), UNION, AS} \) and concatenation via a dot symbol \( \cdot \) (AND), which

![Fig. 1. RDF triple corresponds to the record tuple in RDB.](image)
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 \( \text{WHERE} \) clause to define graph patterns to discover a match for in the query data set. In SPARQL queries, variable names \( V : V_1, ..., V_e \) are prefixed with the question mark ? symbol e.g. \(? V : V_1, ..., V_e\), and value constraints can be defined by the \( \text{FILTER} \) keywords. The SPARQL query offers lots of filtering possibilities: String matching to test strings based on regular expressions \( \text{regex} \) (\( ?V, \text{str}\)), Boolean operators \( (!, \&, \&) \), comparison operators \( (=, \!, >, =, <, <=) \), arithmetic operators \( (*, /, +, -) \), and RDF element operators \( \text{bound}(?V)\), \( \text{isURI} (?V) \) and \( \text{STR}() \). Precisely, a SPARQL graph pattern expression \( GP \) is defined as the following abstract grammar:

\[
\text{Grammar (GP)}: \text{SPARQL graph pattern expression} \\
GP \rightarrow TP \mid GP \land GP \mid \text{OPT} \mid \text{UNION} \mid \text{FILTER} \mid \text{expr}\_cond \\
TP \in (\text{IR} \cup \text{BN} \cup \text{L} \cup \text{V}) \times (\text{IR} \cup \text{L} \cup \text{V})
\]

where, \( \text{FILTER} \) \( \text{expr}\_cond \) represents the \( \text{FILTER} \) construct with a Boolean expression \( \text{expr}\_cond \). Therefore, a SPARQL query defined as:

\[
Q = \text{SELECT} \ ?V_1, ..., ?V_e \ \text{WHERE} \ (GP \ \text{FILTER} \ (\text{expr}\_cond)) \quad ?V_1, ..., ?V_e \in \text{var}(GP)
\]

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

\[
(((? X, \text{name}, ?N) \text{AND} (?X, \text{age}, ?Ag)) \ \text{FILTER} \ (\text{regex}(?N, "mohamed"))
\]

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):**
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:

**Rule 1 (\( TB, \rightarrow CLS, \):** Each table \( TB \) in an RDB unless ISBinarRel(\( TB \)) should be mapped to a class \( CLS, \) in SWO.

### Rule 1 (R_1): \( TB, \rightarrow CLS, \)

<table>
<thead>
<tr>
<th>RDBS</th>
<th>SWOS (RDF/OWL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( TB \land \neg ISBinarRel(( TB )) )</td>
<td>( CLS, i ) is name, ( TB \in RDB, CLS \in SWO. )</td>
</tr>
</tbody>
</table>

This is a basic rule used to identify classes in a clear way from several cases in tables. It depends on the function condition, ISBinarRel(\( TB \)) 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, Rule 1 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 (\( \neg ISBinarRel(\"Stud_\_Cors\") = \text{false} \))

**Rule 2 (\( TB, (A_{\text{type}}) \rightarrow CLS, (DTP_{\text{std}}) \):** Each column \( A_k \in TB \) unless \( pk(A_k,TB) \) or \( \beta k(A_k,TB) \) and \( \neg ISBinarRel(TB) \) should be mapped to a datatype property \( DTP \) in class \( CLS \in SWO \) and datatype of column (type) to XML schema datatype (xsd) of property.

### Rule 2 (R_2): \( TB, (A_{\text{type}}) \rightarrow CLS, (DTP_{\text{std}}) \)

<table>
<thead>
<tr>
<th>RDBS</th>
<th>SWOS (RDF/OWL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( TB(A_{\text{type}}) \land \neg (pk(A_k,TB) \lor \beta k(A_k,TB)) \land \neg ISBinarRel(TB) )</td>
<td>( CLS,(DTP_{\text{std}}) )</td>
</tr>
</tbody>
</table>

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 \( TB, (A_{\text{type}}), pk(A_k,TB), \) \( \beta k(A_k,TB), \) and ISBinarRel(\( TB \)). 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 (xsd datatype corresponding to SQL datatypw).
- All table columns that are ISBinarRel(\( TB \))=true are not mapped to datatype properties.
- Every column unless the predicates \( pk(A_k,TB)=\text{true} \) and \( \beta k(A_k,TB)=\text{true} \) 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 varchar)).

**Rule3** ( pk(Ak,TBi) \rightarrow INVFUNPR(Ak) ): Each attribute Ak in TBi that is pk(Ak,TBi) unless fk(Ak,TBi) should be mapped to both an inverse functional property INVFUNPR(Ak) (owl:InverseFunctionProperty) with a minimum cardinality of 1 (owl:minCardinality) minCRD I(Ak) restriction on the property Ak.

```
| Rule3 ( R.3): pk(Ak,TBi) \rightarrow INVFUNPR(Ak) |
| RDBS | SWOS (RDF/OWL) |
| pk(Ak_{Domain},TBi) \land \neg fk(Ak_{Domain},TBi) \land \neg ISBinaryRel(TBi) | INVFUNPR(I, CLS_{Domain}, CLS_{Inverse_Rng}), minCRD I(A_{inverse}) |
```

This rule used for mapping column primary key to inverse function property INVFUNPR(Ak) and restriction minCardinality constraint of 1 minCRD I(Ak), if the predicate conditions of rules are true. Two cases, unique and 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(Ak) (unique constraint) and to restriction minCRD I(Ak) (not null constraint).

For instance, the primary key Lab_No in table Lab is hold in our example, where pk(Lab_No) \land \neg fk(Lab_No) \land \neg ISBinaryKey(Lab) are true, then INVFUNPR(Lab,No) (Lab_No,Lab_{Domain,xsd^\text{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 TBi,....,TBl \in RDB. Therefore, we will start to map it based on our analysis in Section 3.1.

**Rule4** ( TBi.A \rightarrow \{n\} TBl.B ): Each attribute (column) A in the table TBi that references attribute B in the table TBl should be mapped to an object property (owl:ObjectProperty) OBP that has the source table TBi as its domain (TBi \rightarrow CLS_{Domain}) and destination table TBl as its range (TBl \rightarrow CLS_{Rng}).

```
| Rule4 ( R.4): TBi.A \rightarrow \{n\} TBl,B |
| RDBS | SWOS (RDF/OWL) |
| fk(A, TBi, B, TBl) | OBP(A, CLS_{Domain}, CLS_{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** ( TBi.A \rightarrow \{\text{one:one}\} TBl,B ): If two relations TBi(A1,...,Ai) and TBl(B1,...,Bj) are related to each other through their columns TBi.A \rightarrow \{\text{one:one}\} TBl.B, where fk(A, TBi) references pk(B, TBl), the relation should be 1:1 (TBi.A \rightarrow \{\text{one:one}\} TBl.B). Therefore, fk(A, TBi) is mapped into OBP(A, CLS_{Domain}, CLS_{Rng}) that has the source table TBi as its domain, and the destination table TBl as its range. In the 1:1 relationship, the value \text{TBi.A = TBl.B}. The property is restricted to the same value from the class (TBl) RestOnProp(A, hasValue, TBi). The constraint A \neq null is mapped into a minCardinality in the same restriction, and the property (A) RestOnProp(<A,owl:hasValue, TBi>, <A,owl:minCardinality,xsd^\text{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): TBi.A \rightarrow \{\text{one:one}\} TBl,B |
| RDBS | SWOS (RDF/OWL) |
```

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.Post_N ∧ pk1(Post_No,Postion) ∧ Student.Post_No ≠ null are true. Hence, the OBP(Post_No, Student:domain,Postion:range), 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, A −→ (\text{one-many}) TB, B): If the function IsRelationship(TB, A, TB, B) is true, and TB, A −→ (\text{one-many}) TB, B is false, the relationship is 1: m, and TB, A −→ (\text{many}) TB, B is mapped into an OBP(A,CLS_{\text{Domain}},CLS_{\text{Rng}}). In the 1: m relationship, a column value (TB, B) exists in the column value (TB, A). Therefore, this property is restricted to all values from the class (TB, B) RestOnProp(A,owl : allValueFrom, TB, B). If the constraint A ≠ null holds, it will be mapped into a mincardinality RestOnProp(<A,owl : allValueFrom, TB, B, >, <A,owl : minCardinality, xsd:^\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.

Rule5.3 (TB, A −→ (\text{many}) TB, B, C): 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(A, ..., A) and TB(B, ..., B) are related to each other through the third table TB(C, where pk(C, A, B, TB, C), TB, A −→ \text{\delta} pk(C, A, TB) and TB, B −→ \text{\delta} pk(C, B, TB), then BinaryRel(TB, A, B, TB, C) is hold. In such a situation, only the tables TB(A, ..., A) and TB(B, ..., B) 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_{\text{Domain}},CLS_{\text{Rng}}) and OBP(B,CLS_{\text{Domain}},CLS_{\text{Rng}}) according to the rule of (R_5.2) 1:m relationship.
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 \( (TB,A \rightarrow_{\text{one}}^{\text{many}} TB_{B}^{\text{many}}) \) to map the relationship one:many between TB\(_{A}\) and TB\(_{B}\). Secondly, it calls Rule5.2 \( (TB_{B},B \rightarrow_{\text{one}}^{\text{many}} TB_{A}^{\text{many}}) \) 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. \( NS_{IRI} : \text{http://mo_exp.edu.cn/dbLab/#} \)). The following rules to produces IRIs are:

<table>
<thead>
<tr>
<th>Rule to generate IRI for the class:</th>
<th>Rule to produce IRI for the any property in the class:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (NS_{IRI} + CLS,) )</td>
<td>( CLS,(\text{IRI}) + .&quot;, + A : A \in \text{attr}(CLS,) )</td>
</tr>
<tr>
<td>( CLS,(\text{IRI}) )</td>
<td>( CLS,(A,\text{IRI}) )</td>
</tr>
</tbody>
</table>

Example: \("\text{http://mo_exp.edu.cn/dbLab/#Professor",rdf:type,owl:Class}\) where the \( (\text{http://mo_exp.edu.cn/dbLab/#Professor}) \) is the IRI for the Professor table in our example. Another example, the triple\("\text{http://mo_exp.edu.cn/dbLab/#Professor.Name",rdf:type,owl:datatypeProperty}\), and the triple\( (\text{http://mo_exp.edu.cn/dbLab/#StudentLab_No,rdf:type,owl:objectProperty}) \) 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 \text{Student.Name} \text{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 \( TB_{k} \) is mapped to the class \( CLS,(\text{IRI}) \) then all rows of the table \( l(TB_{k}) = [n_{0},...,n_{n}] \) are transformed to the instance of RDF graphs \( lo(CLS,)_{\text{IRI}} = [gn,...,gn] ; gn = [n_{i},...,n_{t}] , t = (x,p,o) . \). If each column of table \( TB_{k}(A_{1},...,A_{k}) \) transferred to the properties of class \( CLS,(\text{IRI})_{(DTR,(\text{IRI}),...,DTR,(\text{IRI}))} \), then the values of the columns unless null-value in table \( l(TB_{k}) = [n_{0},...,n_{n},...,n_{0},...,n_{n}] \) \( \text{val}(A_{i},n_{0},TB_{k}) \neq \text{null} \) can be mapped to the values of the corresponding property of ontological instance \( lo(CLS,)_{\text{IRI}} = [gn,...,gn,...,gn_{k},...,gn_{k}] . \) Firstly, we initiate a family of predicates that produce \( \text{RowIRI}(\text{RwId}_{\text{IRI}}) \) for the triples being translated according to the following rule.

<table>
<thead>
<tr>
<th>Definition (6): Procedure for generating</th>
<th>( \text{RowIRI}(\text{RwId}_{\text{IRI}}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function header</td>
<td>( \text{RowIRI}(\text{RwId}<em>{\text{IRI}}),TB</em>{k},n_{i} ) )</td>
</tr>
</tbody>
</table>
A RowIRI(RwId_{row}, TB, rw_{row}) generates the identifier RwId_{row} of a row rw_{row} of a relation TB. Thus, given that the facts PK1("Student","Stud_Id") and VALUE("Student","rw1","Stud_Id",1) are hold in our example, the (RwId=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.

<table>
<thead>
<tr>
<th>Rule6:</th>
<th>I(TB), (rw_{1},...,rw_{n}) \rightarrow Io(CL_S,(gr_{1},...,gr_{n}))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Procedure for generating RDF datasets from RDB data.</td>
</tr>
<tr>
<td></td>
<td>I(TB), (rw_{1},...,rw_{n})</td>
</tr>
<tr>
<td></td>
<td>RowIRI(RwId_{row}, TB, rw_{row}), gr_{row}</td>
</tr>
<tr>
<td></td>
<td>rw_{row} \in [{A_{1},...,A_{m}}, TB], \land val(A_{i},rw_{row},TB) \neq null</td>
</tr>
<tr>
<td></td>
<td>\left{ \begin{align*} &amp; t_{4}(RwId_{row}, rdf:type, CL_S) ; \ &amp; t_{5}(RwId_{row}, CLS(A_{i}), getObject(A_{i},rw_{row},TB)) ; \end{align*} \right.</td>
</tr>
</tbody>
</table>

Where, Isfk(A, TB) \leftarrow [(TB, A \rightarrow^{(x)} TB, B), val(A_{i},rw_{row},TB) \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 getObject(A_{i},rw_{row},TB) used to get the object of the triple t_{i} of graph gr_{i} depends on the type of column A_{i} (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}(\{id (NS:Student_1 , rdf:type , NS:Student), r_{1} (NS:Student_1 , NS:Student.Name , Mohamed), r_{2} (NS:Student_1 , NS:Student.Lab_No , NS:Lab_442),...\},...),...),

Lab(gr_{2}(\{id (NS:Lab_442 , rdf:type , NS:Lab), r_{3} (NS:Lab_442 , NS:Lab_Lab_No, NS:Lab_Lab_No,442),...\},...),...). 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.
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 \( Q(\cdot) \), there is SPARQL query \( \sigma^Q(\cdot) \) satisfying the following function:

**Definition (7):** \( Q(\cdot) \rightarrow \sigma^Q(\cdot) \)

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 \( (\sigma) \), Projection \( (\pi) \), Rename \( (\rho) \), Union \( (\cup) \), Difference \( (\setminus) \), Natural Join \( (\bowtie) \), Left Join \( (\leftarrow) \), and Binary Relation.

5.1. Rules for Basic Relational Algebra Operations

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

\[
\sigma_p(TB_1 \times \ldots \times TB_n) : n \geq 0, p \subseteq \{ A, A_1, \ldots, A_n | A \theta \alpha, A \theta \beta, p \phi p_2, IsNull(A), IsNotNull(A), \text{like}*, \text{IN} \}, \theta \in \{ =, \neq, \leq, \geq, > \}, \phi \in \{ \&\& \}, A_i, A_e \in \text{att}(TB), v \text{ is a constant value.}
\]
Where the $p$ stands for an expression condition in the set $\{\theta, \leq, <, \geq, >\}$. $\theta$ is a logical operation \{and, or\} and $p, p_5$ are expiration condition. Therefore, we need to consider all the cases to define a query $Q$ to satisfy the defining condition (7).

<table>
<thead>
<tr>
<th>Rule No.</th>
<th>SQL–Algebra $Q$</th>
<th>SPARQL $Q^\circ$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_7.1</td>
<td>$\sigma_{A=m}(TB)$</td>
<td>$(GP \text{ FILTER } (?A = v))$ or $(GP \text{ FILTER } \text{regex}(?A, v))$</td>
</tr>
<tr>
<td>R_7.2</td>
<td>$\sigma_{A=1} \land \sigma_{A=2} \lor \sigma_{A=3}(TB)$</td>
<td>$(GP \text{ FILTER } (?A=1 \land \sigma_v 2 \lor \sigma_v 3 \text{ between } 01-01-2018 \text{ and } 10-01-2018))$</td>
</tr>
<tr>
<td>R_7.3</td>
<td>$\sigma_{\text{between}A}(TB)$</td>
<td>$(GP \text{ FILTER } (\text{bound}(?A)))$</td>
</tr>
<tr>
<td>R_7.4</td>
<td>$\sigma_{\text{between}A}(TB)$</td>
<td>$(GP \text{ FILTER } (\text{bound}(?A)))$</td>
</tr>
<tr>
<td>R_7.5</td>
<td>$\sigma_{A \text{ in } (v_1, ..., v_n)}(TB)$</td>
<td>$(GP \text{ FILTER } (\text{regex(?A, v)}))$</td>
</tr>
<tr>
<td>R_7.6</td>
<td>$\sigma_{A \text{ in } (v_1, ..., v_n)}(TB)$</td>
<td>$(GP \text{ FILTER } (\text{regex(?A, v)})$</td>
</tr>
</tbody>
</table>

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.

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

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 = "Zhang" constraints respectively. More details are shown in Fig. 6 in equivalent SPARQL query $(Q^\circ)$.

```
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. $Q^\circ$ selection operation algebra $\sigma_{\text{Name}="Zhang"}(\text{Professor})$

where $a$ 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 [1]):** This rule used to rewrite a SPARQL query $(Q^\circ)$ corresponding to relational algebra $(Q)$ operation (projection [1]), 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:

<table>
<thead>
<tr>
<th>Rule8 (R_8): Projection [1]</th>
<th>SQL–Algebra $Q$</th>
<th>SPARQL $Q^\circ$</th>
</tr>
</thead>
</table>

This query involves three triple patterns (TPs) concatenation via AND (\(\cap\)) located at the end of triple pattern. The result set is restricted to objects of the NS:Lab class in the first TP1 \(\{?x \text{ a NS:Lab}\}\) and assigned to the variable \(?x\). The variable \(?x\) has the same values that match each triple patterns (TP2 and TP3). The TP2 \(\{?x \text{ NS:Lab:Lab_No ?Lab_No}\}\) 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\(\{?x \text{ NS:Lab:Lab_Name ?Lab_Name}\}\) 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 \(\rho\)):** In the in relational algebra, a rename \(\rho\) 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:

\[
\text{Rule9 (R_9): Rename } \rho
\]

<table>
<thead>
<tr>
<th>SQL-Algebra</th>
<th>SPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\prod_{TB.A_1, ..., TB.A_n} (TB))</td>
<td>Select ({TB.A_1, ..., TB.A_n } \text{ where } {GP}) order by ((\text{ASC}</td>
</tr>
</tbody>
</table>

Obviously, the SPARQL expression \(Q^o\) correspond to relational algebra \(Q\), that because the rename operator in \(Q\) and \(Q^o\) 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 \(\{\text{11}\}\)-Select statements. Then the equivalent rule query \(Q\) to satisfy the defining condition (7) can be defined as:

\[
\text{Rule10 (R_10): Union } \cup
\]

<table>
<thead>
<tr>
<th>SQL-Algebra</th>
<th>SPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\prod_{TB.A_1, ..., TB.A_n} (TB_1) \cup \prod_{TB.A_1, ..., TB.A_n} (TB_2))</td>
<td>Select ({\text{where } {\text{Select } ?TB.A_1, ..., TB.A_n \text{ where } {GP}} \text{ UNION } _UNION} \text{ where } {GP}})</td>
</tr>
</tbody>
</table>

It can be noticed that the \(Q\) expression unifies all rows from the attributes \((A_{1_1}, A_{n})\) in \(TB_1\) to \(TB_2\) relation. The first requirement is to perform the projection within the UNION clause to restrict the \((A_{1_1}, A_{n})\) variables for both \((A_{1_1}, A_{n})\) attributes. Therefore, the SPARQL query \(Q^o\) will return all values \((A_{1_1}, A_{n})\) that originated in \(TB_1\) to \(TB_2\).

**Rule11 (Difference \(\setminus\)):** A difference \(\setminus\) 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:

\[
\text{Rule11 (R_11): Difference } \setminus
\]

<table>
<thead>
<tr>
<th>SQL-Algebra</th>
<th>SPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\prod_{TB.A_1, ..., TB.A_n} (TB_1) \setminus \prod_{TB.A_1, ..., TB.A_n} (TB_2))</td>
<td>Select ({A_{1_1}, ..., A_{n} } \text{ where } {\text{where } {\text{?x a NS:TB_1. A_1 XS:TB_1. A_2 ... FILTER NOT EXISTS } {\text{?x a NS:TB_2. A_1 XS:TB_2. A_2 ...}}}})</td>
</tr>
</tbody>
</table>

The SPARQL query and its result are represented in the following example (Fig. 7):
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 ?Lab_No} 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 ?Lab_No} 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 ?Lab_No} 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\^\circ 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 \(\times\)):** It is a binary operator, which used to combine tuples from two/more tables \(TB_1 \times TB_2 \times \ldots \times TB_n\) through on a common field between them. The result of \(TB_1 \times TB_2\) is a set of all combination tuples 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 (corresponding to FKs in RDB). Then the equivalent rule query \(Q^\circ\) to satisfy the defining condition (7) can be defined as:

**Rule12.1 (R_12.1): (Natural join \(\times\))**

<table>
<thead>
<tr>
<th>SQL-Algebra Q</th>
<th>SPARQL (Q^\circ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\prod TB_1.A_1, TB_2.A_2, \ldots, TB_{n-1}.A_{n-1}, TB_n.A_n (TB_1 \times TB_2 \times \ldots \times TB_n))</td>
<td>Select ?x, ?y, ?z, ?y.a NS:Student, ?y.n NS:Lab, ?y.n.a NS:Lab, ?z.a NS:Student, ?z.n NS:Lab, ?z.n.a NS:Lab, Optional {?y.n.a NS:Lab, ?x.n.a NS:Lab} Where {?x = ?y.n.a, ?z = ?y.n.a, ?y = ?z.n.a, ?z.n.a = ?z.n.a}</td>
</tr>
<tr>
<td>Where (\text{TB}_1.A_1) and (\text{TB}_2.A_2) are common attributes.</td>
<td>Optional ((TB_1.A_1)) Optional ((TB_2.A_2))</td>
</tr>
</tbody>
</table>

The graph pattern GP \{?x NS:TB_1.A_1 ?TB_1.A_2 ?TB_1.A_3 a NS:TB_2\} has an object property (NS:TB_1.A_1) represented by the variable ?TB_1.A_1 which used to link between two classes NS:TB_1 and NS:TB_2. To ensure no any data base of triples, the OPTIMAL operator for all properties of class was used except on the common property that should be its value is not-nil. If we change the rule condition by BOUND operator in part (\(\text{OPTIMAL} (\{?x a NS:TB_1.A_1\})\) to \(\{?x a NS:TB_1.A_1\}\)) the same result was observed. The following Q2 example with it equivalent SPARQL (\(Q^\circ_2\)) and its result (Fig. 8) that is generated with input:

\[Q_2: \prod_{\text{Student.Stud_Id, Student.Name, Lab.Lab_Name} (\text{Student} \times \text{Lab})}\]
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 $\bowtie$): To satisfy the defining condition (7) according to left join, the equivalent query $Q$ can be defined as follows:

$$
\begin{align*}
&\text{SQL–Algebra } Q \\
&TB_1.A \text{ and } TB_2.A_4 \text{ are common columns in this expression}
\end{align*}
$$

The different between this rule and Rule 12.1 is the use of nested OPTIONAL. The condition graph pattern GP {?x NS:Student.Lab_No ?lab_no} 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 $Q^0$ from a Binary Relation

Assume that $Q$ is a query algebra over binary relation $\text{BinaryRel}(TB_3.A,B,TB_1.A,TB_2.B)$ and $\text{ISBinaryRel}(TB_3)$ holds, based on our rules (R.5.3 and R.6). These rules make the property $A$ be part of the class $TB_2$, and property $B$ is part of the class $TB_1$, where $A$ and $B$ refers to its class (Fig. 5). The binary

$$
\begin{align*}
&\text{I}_{\text{Q}_3} \\
&\prod_{\text{Student.Stude_Id}, \text{Student.Name}, \text{Lab.Lab_Name}} (\text{Student} \bowtie \text{Lab})
\end{align*}
$$

The condition GP {?x NS:StudentLab_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).
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 (TB1 and TB2) 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**

<table>
<thead>
<tr>
<th>SQL–Algebra Q</th>
<th>SPARQL Q°</th>
</tr>
</thead>
</table>

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.


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 (NetBeans 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 MYSQL 5.6 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 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
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 (Q₁) corresponding to the query in relational algebra (Q₀). Query (Q₀) 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 (Q6) using our system interface (Fig. 13), which used as the interfaces to execute a SPARQL query on RDF triple s for querying generated ontologies. From Fig. 14 and 15 it can be observed that the returned dataset of Q5 and its equivalent (Q6) are same and this reflects the validity of our approach.

\[
P_5 \left( \text{Student} \times \text{Lab} \times \text{Professor} \right)
\]

\[
Q_6 \left( \text{Student} \cup \text{Lab} \cup \text{Professor} \right)
\]

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 SPARQL( Q6) query corresponding to the operations of SQL algebra (projection, rename, and union).

\[
P_6 \left( \text{Student} \cup \text{Lab} \cup \text{Professor} \right)
\]

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 returned 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

<table>
<thead>
<tr>
<th>Tables</th>
<th>Rows</th>
<th>Columns have null-value</th>
<th>Rows have null-value</th>
<th>SPARQL TP used to return class tuples</th>
<th>Returned tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>70000</td>
<td>Lab_No=7614 Age=2500</td>
<td>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.</td>
<td>{?x a NS:Student}</td>
<td>70000</td>
</tr>
<tr>
<td>Lab</td>
<td>3000</td>
<td>Lab_Name=199 Prof_No=111</td>
<td>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.</td>
<td>{?x a NS:Lab}</td>
<td>3000</td>
</tr>
<tr>
<td>Courses</td>
<td>100</td>
<td></td>
<td></td>
<td>{?x a NS:Courses}</td>
<td>100</td>
</tr>
<tr>
<td>Stud_Cors</td>
<td>27000</td>
<td></td>
<td></td>
<td>Q4</td>
<td>27000</td>
</tr>
<tr>
<td>Professor</td>
<td>100</td>
<td></td>
<td></td>
<td>{?x a NS:Professor}</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2. SPARQL (Q°) Queries

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

Table 3. Results of Queries and their Executing Time

<table>
<thead>
<tr>
<th>Q</th>
<th>Q Rows</th>
<th>Time (s)</th>
<th>Q° Tuples</th>
<th>Time (s)</th>
<th>Q° Tuples</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>375</td>
<td>0.007</td>
<td>375</td>
<td>0.008</td>
<td>366</td>
<td>0.009</td>
</tr>
<tr>
<td>Q2</td>
<td>62386</td>
<td>0.046</td>
<td>62386</td>
<td>0.06</td>
<td>54513</td>
<td>0.095</td>
</tr>
<tr>
<td>Q3</td>
<td>70000</td>
<td>0.071</td>
<td>70000</td>
<td>0.075</td>
<td>62386</td>
<td>0.1</td>
</tr>
<tr>
<td>Q4</td>
<td>27000</td>
<td>0.025</td>
<td>27000</td>
<td>0.035</td>
<td>24190</td>
<td>0.04</td>
</tr>
<tr>
<td>Q5</td>
<td>70000</td>
<td>0.07</td>
<td>70000</td>
<td>0.075</td>
<td>58174</td>
<td>0.095</td>
</tr>
</tbody>
</table>
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 a set of formal rules to rewrite the SPARQL query corresponding to 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 to 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 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Model</th>
<th>Analysis concepts of RDB and Ont.</th>
<th>Rel. Schema</th>
<th>Rel. Data</th>
<th>Null value</th>
<th>Data Transform</th>
<th>Generate Formal/Informal Identifier</th>
<th>Validation/Semantic query</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buccella et al.</td>
<td>Semi-autoNo</td>
<td>No</td>
<td>M:N</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Formal/Expository</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Stojanovic et al</td>
<td>Semi-auto</td>
<td>Yes</td>
<td>1:1, 1:M</td>
<td>1:M</td>
<td>No</td>
<td>No</td>
<td>Formal</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Astrova et al.</td>
<td>Auto</td>
<td>No</td>
<td>1:1</td>
<td>No</td>
<td>No</td>
<td>Yes(weak)</td>
<td>Informal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Shen et al.</td>
<td>Semi-auto</td>
<td>Yes</td>
<td>1:M</td>
<td>1:M</td>
<td>No</td>
<td>No</td>
<td>Informal</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Li et al.</td>
<td>Auto</td>
<td>Ontology</td>
<td>M:N</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Informal</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Mohamed et al.</td>
<td>Auto</td>
<td>No</td>
<td>M:N</td>
<td>No</td>
<td>No</td>
<td>Yes(weak)</td>
<td>Informal</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Zhang &amp; Li</td>
<td>Semi-auto</td>
<td>Yes(Weak)</td>
<td>1:1</td>
<td>No</td>
<td>No</td>
<td>Yes(weak)</td>
<td>Informal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Bakkas &amp; Bahaj</td>
<td>Auto</td>
<td>No</td>
<td>M:N</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Informal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Proposed approach</td>
<td>Auto</td>
<td>Yes</td>
<td>1:1, 1:M, 1:1, 1:M, M:N</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Formal</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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-formed 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 these 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


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).