A Novel and Complete Approach for Storing RDF(S) in Relational Databases

Fu Zhang[1], Qiang Tong[2], and Jingwei Cheng[1]
[1]School of Computer Science and Engineering, Northeastern University, China
[2]School of Software, Northeastern University, China

Abstract: *Resource Description Framework (RDF) and RDF Schema (collectively called RDF(S)) are the normative language to describe the Web resource information. With the massive growth of RDF(S) information, how to effectively store them is becoming an important research issue. By analysing the characteristics of RDF(S) data and schema semantic information in depth, this paper proposes a multiple storage model of RDF(S) based on relational databases. An overall storage framework, some detailed storage rules, a storage algorithm and a storage example are proposed. Also, the correctness of the storage approach is discussed and proved. Based on the proposed storage approach, a prototype storage tool is implemented, and experiments show that the approach and the tool are feasible.*

Keywords: *RDF, RDF schema, relational database, storage.*

Received June 26, 2016; accepted October 11, 2017

1. Introduction

Resource Description Framework (RDF) and RDF vocabulary description language RDF Schema (collectively called RDF(S)) are the normative language to describe the Web resource information [20]. By means of RDF(S), the information can be easier to share, protect, and retrieve in the Web. In current, the formal acceptance of RDF(S) by W3C stimulates their utilization in many areas (e.g., life sciences, GIS, Semantic Web, and etc.,) [12]. With the massive growth of RDF(S) information, how to effectively store them is becoming an important research issue.

To this end, many approaches have been developed to store RDF(S) (see surveys [9, 14, 17]). In general, the RDF(S) storage methods can be classified into several main categories: based on the file system, the special storage tools, and the databases. The first category methods store RDF(S) in XML/RDF format files, the second category methods store RDF(S) in the special storage tools, and the third category methods store RDF(S) in the database systems. Moreover, some RDF middlewares and parsers such as Jena and Sesame can be used to implement the access to the physical RDF data store, read and parse the RDF statements.

Among the storage methods above, the storage of RDF(S) based on database systems occupy very important position. As we have known, the database research community has successfully developed a wide theory corpus and a mature and efficient technology to deal with large and persistent amounts of information. In particular, relational databases have mature theory and products. RDF(S) stores, which are backed by relational databases, can apply different kinds of storage models for representing the RDF(S) in the underlying relational schemas. In this case, the storage and retrieval functionality of existing relational database management systems can be fully utilize.

As the literatures [1, 5] showed that, there are several different RDF(S) storage patterns based on relational databases, e.g., the common Horizontal [2, 5], Generic/Vertical [6, 10, 15, 23], and Specific/ Binary patterns [3, 4, 18], however, RDF(S) information in some real applications is different in respects of scales and characteristics. Therefore, it is difficult to give a unified pattern which is enough to effectively store all RDF(S) information. Besides, we found that many existing methods (e.g., [2, 4, 18, 23]) only focused on storing the RDF data and did not fully consider the semantic information storage of RDF Schema corresponding to the RDF data. To this end, in this paper we proposes a multiple storage model of RDF(S) based on relational databases by analysing the semantic characteristics of RDF data and RDF Schema in depth. In brief, the paper makes the following main contributions:

- After analysing semantic characteristics of RDF and RDF Schema, we propose an overall architecture of storing RDF(S) in relational databases.
- Based on the architecture, we further propose storage rules and explain how to store RDF(S) in relational databases with a running example in detail. Also, the correctness and quality of the storage approach are proved and analysed.
- Finally, on the basis of the proposed approach, we give a storage algorithm and test and compare our approach with the existing work. The storage and

query examples and the comparison results show that the approach is feasible and efficient.

The remainder of this paper is organized: section 2 introduces basic concepts. Section 3 proposes a multiple storage model of RDF(S) based on relational databases. Section 4 implements a prototype storage system. Section 5 introduces related work. Section 6 shows conclusions and future work.

2. Preliminaries

In this section, some preliminaries on RDF(S) and relational databases are recalled.

2.1. RDF(S)

RDF [20] is a framework for expressing the Web resource information. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. The basic idea of RDF is: Anything is called "resource". A resource can be identified by URI (Universal Resource Identifier). A resource may have some "properties", and these properties may have "values", which may be literal values (e.g., string or float) or other resources. The relationships among resources, properties and values can be described by "statements", which always have the structure of triple: <subject predicate object>.

But RDF cannot define semantic information, e.g., RDF cannot state http://www.example.org/brotherof can be used as a property and that its subjects and objects of triples must be the resources of the class http://www.example.org/Person, which can be described by RDF vocabulary Description Language RDF Schema [20]. It uses the notion of "class" to specify categories that can be used to classify resources. The relation between an instance and its class is stated through the "type" property. With RDF Schema one can create hierarchies of classes and "sub-classes" and of properties and "sub-properties". Type restrictions on the subjects and objects of particular triples can be defined through "domain" and "range" restrictions. An example of a domain restriction was given above: subjects of "brotherOf" triples should be of class "Person". Also, one can define a class "Faculty" is a subclass of the class "Staff", and "John" is an instance of "Faculty".

In this paper, RDF and RDF Schema are collectively called RDF(S). In brief, an RDF(S) model R can be represented as $R = (R_I, R_T)$, where $R_I = C \cup P \cup D \cup I$ is a set of URIs partitioned into a set C of class identifiers, a set P of property identifiers, a set D of datatype identifiers, and a set $\mathcal{I}$ of individual identifiers; R_T is a set of triples defined over R_I.

2.2. Relational Databases

The relational database was first defined in June 1970 by Codd [8] and has become the predominant type of databases. In general, each database is a collection of tables, which are called relations, hence the name "relational database". A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an instance and its information.

Formally, a relation is usually described as a table, which is organized into rows and columns. A domain describes the set of possible values for a given attribute, and can be considered a constraint on the value of the attribute. Moreover, the keys within a database are used to define the relationships among the tables. A primary key uniquely specifies a tuple within a table. When a primary key migrates to another table, it becomes a foreign key in the other table. A foreign key is a field in a relational table that matches the primary key column of another table. In addition, there are several constraints in the relational databases, e.g., entity integrity constraints, i.e., every relation should have a primary key and the value of the primary key in each tuple should be sole and cannot be null; referential integrity constraints, i.e., let a relation r have a foreign key FK and the foreign key value of a tuple t in r be t[FK], and let FK quote the primary key PK of relation r′ and t′ be a tuple in r′, then referential integrity constraint demands that t[FK] comply with the constraint: t[FK] = t′[PK]/NULL. The applications access data in relational databases by specifying queries, which use operations such as select to identify tuples, project to identify attributes, and join to combine relations. Relations can be modified using the insert, delete, and update operators.

3. Storing RDF(S) in Relational Database

The section proposes a multiple storage model of RDF(S) based on relational databases by analysing the semantics of RDF(S), including:

1. We propose an overall architecture of storing RDF(S) in relational databases (section 3.1).
2. Based on the architecture, we further propose storage rules and explain with a running example (section 3.2).
3. The correctness of the storage approach is proved (section 3.3).

3.1. An Overall Storage Architecture

In the following we propose an overall architecture of storage approach, which is helpful to well understand the storage process of RDF(S). Figure 1 shows an overall architecture of storage approach. The analyses and introduction of each table are explained as follows:

- *Resource and Namespace tables*: As mentioned in

Section 2, RDF(S) resources are expressed by URIs (Universal Resource Identifiers). Each URI includes a namespace and its resource name. For example, http://purl.org/dc/elements/1.1/creator is an URI of a resource, where http://purl.org/dc/ elements/1.1/ is a namespace, and creator is the name of a term. Many resources have the same namespace. Therefore, resource and namespace tables need to be created to save storage space.

- *Class table*: In real-world applications, many instances having the same properties are gathered into classes. The numbers of classes may be less than the numbers of instances and properties. Therefore, a class table is created for each class. Moreover, when the value of a property belonging to that class is literal (e.g., string, integer, and decimal), the property is inserted as a column into the class table. In this case, it does not need to create tables for each property.
- *Class hierarchy table*: A class hierarchy table is created to store all of RDF(S) subclass/superclass (i.e., rdfs:subClassOf) relations.
- *Property field table*: In RDF(S), many properties contain the constraints of domain and range. Therefore, instead of creating tables for each property, only a property field table is created to store all of the constraints.
- *Property hierarchy table*: A property hierarchy table stores all of RDF(S) subproperty/superproperty (i.e., rdfs:subPropertyOf) relations.
- *Property relation table*: When the value of a property of a resource is another resource (non-literal), the property expresses the relationship between two resources, i.e., there is a foreign key reference between two resources. In order to reduce the connections of properties among tables, the property relation table is created for each such property to store relationship between resources.
- *Multi-Property table*: In RDF(S), a property may be a multi-valued property. In relational databases, the multi-valued properties cannot be directly stored. Therefore, a multi-valued property table is created to store the multi-valued properties.

3.2. The Detailed Storage Rules

Based on the storage framework in section 3.1, in the following we further give some detailed storage rules, which are illustrated by means of an example taken from the education domain.

Figure 2 shows an RDF(S) model (including RDF Schema information and RDF instance data). Here, for ease of understanding, the graphical structure is used to describe the RDF(S) model and parts of properties in RDF Schema are omitted.

Given an RDF(S) model $R = (R_I, R_T)$ as mentioned in section 2.1, the following rules introduce how to store the RDF(S) model R in a relational database.

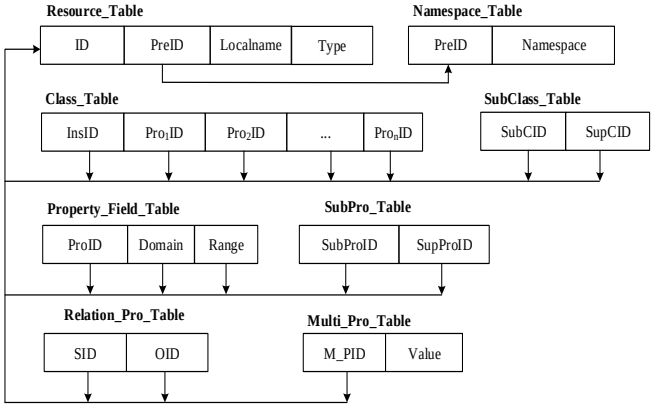


Figure 1. An overall architecture of storing RDF(S) based on relational databases.

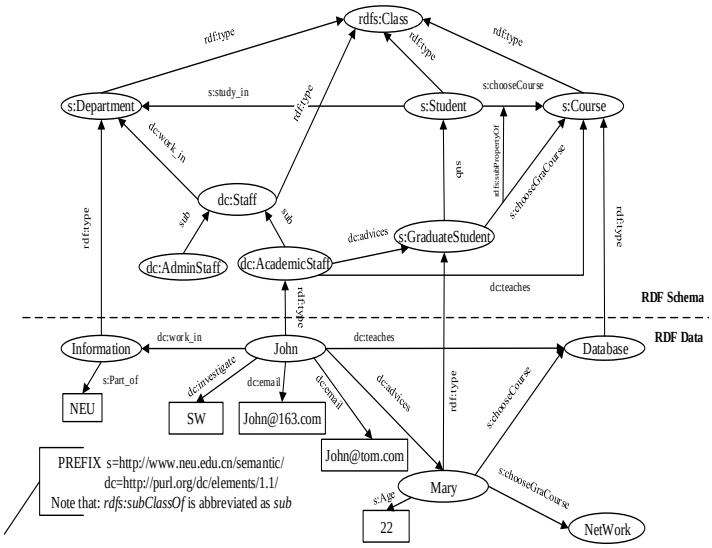


Figure 2. An RDF(S) modelling parts of the reality at a university.

- *Rule* 1 (Storage of RDF(S) resources): Given the set of resource identifiers R_I of RDF(S) R, creating two tables named Resource_Table and Namespace_Table in Figure 1.

In detail, the Namespace_Table contains 2 fields (PreID and Namespace), where PreID is the primary key of the table, which uniquely identifies a namespace; the Resource_Table contains 4 fields (ID, PreID, Localname and Type), where ID is the primary key, which uniquely identifies a resource, PreID and LocalName together describe a resource IRI (International Resource Identifier), and Type identifies the type of a resource, i.e., class, property or individual.

For example, Table 1 stores all of the resource information of RDF(S) in Figure 2, including classes, properties, and individuals.

Table 1. Resource_table and namespace_table.

Namespace_table.

PreID	Namespace
Pre_1	http://www.neu.edu.cn/semantic/
Pre_2	http://purl.org/dc/elements/1.1/

Resource_table.

ID	PreID	Localname	Type
c_1	Pre_1	Department	Class
c_2	Pre_2	Staff	Class
c_3	Pre_2	AdminStaff	Class
c_4	Pre_2	AcademicStaff	Class
c_5	Pre_1	Student	Class
c_6	Pre_1	GraduateStudent	Class
c_7	Pre_1	Department	Class
c_8	Pre_1	Course	Class
p_1	Pre_1	study_in	Property
p_2	Pre_2	work_in	Property
p_3	Pre_1	chooseCourse	Property
p_4	Pre_2	advices	Property
…	…	…	…
i_1	Pre_2	John	Individual
i_2	Pre_1	Mary	Individual

- *Rule* 2 (Storage of RDF(S) classes): Given a class $c \in C$ in RDF(S), creating a class table named *c*_Table, and inserting the literal properties as the columns into the class table as mentioned in section 3.1.

In detail, each class table may contain many individual instances, and they are identified by the primary key InsID in the class table (note that InsID reference to the key ID in Namespace_Table as shown in Figure 1). In addition, when the value of a property belonging to the class c is literal (e.g., string, integer, and decimal), it does not need to create tables for each such property. The literal properties are inserted as the columns into the class table c_Table, and each property $p_i \in P$ are identified by Pro_iID, which reference to the key ID in Namespace_Table as shown in Figure 1.

For example, Table 2 stores the class AcademicStaff, its individual instance John (i_1), and a property investigate (p_7) and a multi-valued property email (p_8) in the RDF(S) of Figure 2. Noted that, M_PID_1 is used to identify values of the multi-valued property email, which will be stored in a multi-valued property table as will be introduced in later Rule 7. The other classes in Figure 2 can be stored similarly.

Table 2. Academic staff_table.

InsID	p_7	p_8
i_1	SW	M_PID_1

- *Rule* 3 (Storage of RDF(S) class hierarchies): Given all of the class hierarchy relations $<c_i$ rdfs:subClassOf $c_j>$ in RDF(S), where $c_i, c_j \in C$, and $i \neq j$, creating a class hierarchy table SubClass_Table to store all of the class hierarchy relations.

In detail, SubClass_Table contains two fields (i.e., SubCID and SupCID), used to represent the subclasses and superclasses of the class hierarchy relations.

For example, Table 3 stores all of the class hierarchies of RDF(S) model in Figure 2, including AdminStaff (c_3) is a subclass of Staff (c_2), AcademicStaff (c_4) is a subclass of Staff, and GraduateStudent (c_6) is a subclass of Student (c_5).

Table 3. Sub class_table.

SubID	SupCID
c_3	c_2
c_4	c_2
c_6	c_5

- *Rule* 4 (Storage of RDF(S) property fields): Given properties and the constraints of their domains and ranges, creating a property field table named Property_Field_Table, used to store all property fields.

In detail, the table Property_Field_Table contains three fields (i.e., ProID, Domain, and Range), used to represent the domain and range of a property.

For example, Table 4 stores all of properties and their domains and ranges in RDF(S) model of Figure 2.

Table 4. Property_field_table.

ProID	Domain	Range
p_1	c_5	c_7
p_2	c_2	c_7
p_3	c_5	c_8
p_4	c_4	c_6
p_5	c_6	c_8
…	…	…

- Rule 5 (Storage of RDF(S) property hierarchies): Given all of the property hierarchy relations $<p_i$ rdfs:subPropertyOf $p_j>$ in RDF(S), where $p_i, p_j \in P$, and $i \neq j$, creating a property hierarchy table SubPro_Table to store all property hierarchy relations.

In detail, the table SubPro_Table contains two fields (SubProID and SupProID), used to represent the subproperties and superproperties of the property hierarchy relations, respectively.

For example, Table 5 stores the property hierarchies, including chooseGraCourse (p_5) is a subproperty of ChooseCourse (p_3), where p_i is the property identifier as shown in Table 1.

Table 5. SubPro_Table.

SubProID	SupProID
p_5	p_3

- *Rule* 6 (Storage of RDF(S) property relations): Given a property $p \in \mathcal{R}$ in RDF(S), if the value of the property is non-literal value, the property expresses the relationship between resources,

creating a property relation table named Relation_*p*_Table for *p*.

In detail, Relation_*p*_Table contains two fields (i.e., *SID* and *OID*), used to represent *subjects* and *objects* of the property *p*. Here we create a property table for each non-literal property and such table can intuitively reflect the property relation.

For example, Table 6 stores the property relation *advices* (*p_4*) in Figure 2, the property *advices* represents the relationship between subject *John* (*i_1*) and object *Mary* (*i_2*).

Table 6. Relation_advices_Table.

SID	OID
i_1	i_2

- *Rule* 7 (Storage of RDF(S) multi-valued properties): Given multi-valued properties $p \in \mathcal{R}$ in RDF(S), creating a property table Multi_Pro_Table.

In detail, the table Multi_Pro_Table contains two fields (i.e., M_PID and value), used to store the multi-valued properties and their values.

For example, Table 7 stores the multi-valued property *email* in Figure 2, where M_PID_1 is used to identify the values of *email* as shown in Table 2.

Table 7. Multi_Pro_Table.

M_PID	Value
M_PID_1	John@163.com
M_PID_1	John@tom.com

- *Rule* 8 (Storage of RDF(S) datatypes): Given datatypes of RDF(S) (i.e., XML Schema datatypes [24]), and they will be mapped to the corresponding SQL datatypes as shown in Table 8.

Table 8. Mapping RDF(S) datatypes to SQL datatypes.

RDF(S) datatypes	SQL datatypes
xsd:integer	INTEGER
xsd:decimal	DECIMAL
xsd:float	FLOAT
xsd:Date	DATE
...	...

3.3. The Correctness of Storage

Based on the approach proposed in Sections 3.1 and 3.2, RDF(S) can be stored in relational databases. From the view of the storage procedures, it shows that the storage approach can be seen as a transformation. There is no a standard that can be used to prove the correctness of the transformation between two kinds of models. As the literature [16, 22], pointed out, if a transformation can keep the information capacity, then it can be considered as the correct transformation. Here, based on the information capacity theory [16, 22], the following Theorem 1 proofs the correctness of the storage approach by proving that the stored procedures can keep the information capacity.

- *Theorem* 1: Given a RDF(S) model *R*, $\varphi(R)$ is the corresponding relational database based on the approach above, if φ is an injective function from *R* to $\varphi(R)$, then the storage is information capacity preserving storage.
- Proof (sketch). Assuming that $\tau \in R$ is an individual instance of a class $c \in C$ in RDF(S) model R, then $\varphi(\tau)$ is a tuple in the stored class table c_Table as mentioned in Section 3.2. Formally, φ can be defined as follows (Here, we take the class table as example): if m classes $\{c_1, \ldots, c_m\} \in C$, each class c_i contains n literal properties $\{p_i^1, p_i^2, \ldots, p_i^n\}$ and s individual instances $\{id_i^1, id_i^2, \ldots, id_i^s\}$, then the following three mapping relations can be established:

1. $\varphi(\tau)[InsID_i^k] \leftarrow id_i^k$.
2. $\varphi(\tau)[ProID_i^j] \leftarrow p_i^j$.
3. $\varphi(\tau)[value_i^j] \leftarrow \tau[p_i^j]$.

Based on the mapping relations above, the RDF(S) model R can be stored into the corresponding relational database $\varphi(R)$. Next, we prove that φ is an injective function. Assuming $\tau_1 = (\tau_1[p_i^1], \tau_1[p_i^2], \ldots, \tau_1[p_i^n])$ and $\tau_2 = (\tau_2[p_i^1], \tau_2[p_i^2], \ldots, \tau_2[p_i^n])$ are two different instances of class c_i in R, then according to the definition of φ above, there are two corresponding tuples in the stored table: $\varphi(\tau_1) = (\varphi(\tau_1)[IndID_i^1], \varphi(\tau_1)[ProID_i^j], \varphi(\tau_1)[value_i^j])$ and $\varphi(\tau_2) = (\varphi(\tau_2)[IndID_i^2], \varphi(\tau_2)[ProID_i^j], \varphi(\tau_2)[value_i^j])$ such that $\varphi(\tau_1) \neq \varphi(\tau_2)$, where $j \in \{1, \ldots, n\}$, that is to say, there is at least one $j \in \{1, \ldots, n\}$ that makes $\tau_1[p_i^j] \neq \tau_2[p_i^j]$, and thus φ is an injective function. As a result, it can be inferred that the transformation from the RDF(S) model R to the relational database $\varphi(R)$ is information capacity retentive, i.e., the storage is an information capacity preserving and correct storage.

4. Prototype Storage Tool and Experiments

4.1. Prototype Storage Tool

On the basis of the proposed storage approach in the previous sections, we implemented a prototype tool called RDFS2RDB for storing RDF(S) in relational databases. The tool takes RDF(S) data sets as input and the stored relational databases as output. The following briefly describes the overall framework, gives a running example, and further tests and compares the approach and tool with the common existing works.

RDFS2RDB is developed by Java language on MyEclipse 7.0 platform. In the implementation, the java.awt and javax.Swing packages are used to exploit the graphical user interface, and the results are stored in MySQL. The overall framework mainly includes: an parse module, which uses Jena API [11] and the SPARQL query language [21] to parse the RDF/XML data sets, and also some semantic information of RDF

Schema will be analysed and extracted as mentioned in section 3; and a storage module, this module takes the parsed results as input, and then according to the Algorithm 1, the data and schema information of RDF(S) data sets are stored in relational databases. The algorithm 1 is the straightforward consequences of the storage approach proposed in section 3, and thus the detailed discussion of the algorithm is omitted. Moreover, all of the input RDF(S), the parsed results, and the stored results are displayed on the graphical user interface.

Here, we give the screen snapshot of RDFS2RDB, and an example is provided to well show the running process. Figure 3 shows the screen snapshot of RDFS2RDB, which displays the storage of an RDF(S) data set (including the RDF data and RDF Schema information in Figure 2) in a relational database. In Figure 3, the source RDF(S) information, the parsed results, and the target database information are displayed in the left, middle and right areas, respectively.

Algorithm 1： The storage algorithm of RDF(S) in relational databases

Noted that the algorithm is given according to the approach in Section 3, and the algorithm only provides the storage steps of some main classes and properties.

Input: RDF(S) Model R
Output: Relational Tables
(1) Begin:
(2) CreateResourceTable(R);//Create the corresponding resource and namespace tables as shown in rule 1
(3) EnQueue(Q, rdfs:Class); // Push the classes into queue Q
(4) while(!isEmpty(Q))
(5) C = DeQueue(Q) // Remove an element from the queue Q and delete it from the queue
(6) if (C != "rdfs:Class")
(7) According to the rule 2, store the class C to relational database
(8) if (hasChildNodes (C)) // To determine whether a node C has a child node
(9) for each $C_i \in SubClassOnNextLevel(C)$ (i = 1…n) // All subclasses of the class C
(10) EnQueue(Q, C_i). According to the rule 3, store the corresponding class hierarchies
(11) end for
(12) EnQueue(Q, rdf:Property) // Push the properties into the queue
(13) while(!isEmpty(Q))
(14) if ((P=DeQueue(Q)) != "rdf:Property")
(15) if (range(P=DeQueue(Q)) = Literal)
(16) According to rules 2 and 7, insert the property into class table and create multi-valued property table
(17) else According to the rule 6, create the corresponding tables
(18) if (hasChildNodes(P))
(19) for each $P_i \in SubPropertyOnNextLevel$ (P) (i = 1…n) // All subproperties of P
(20) EnQueue(Q, P_i). According to the rule 5, store the corresponding property hierarchies
(21) end for
(22) if (!isEmpty(Pro_Fields)) // There are constraints of properties (i.e., domains and ranges)
(23) According to the rule 4, store the corresponding property constraints
End

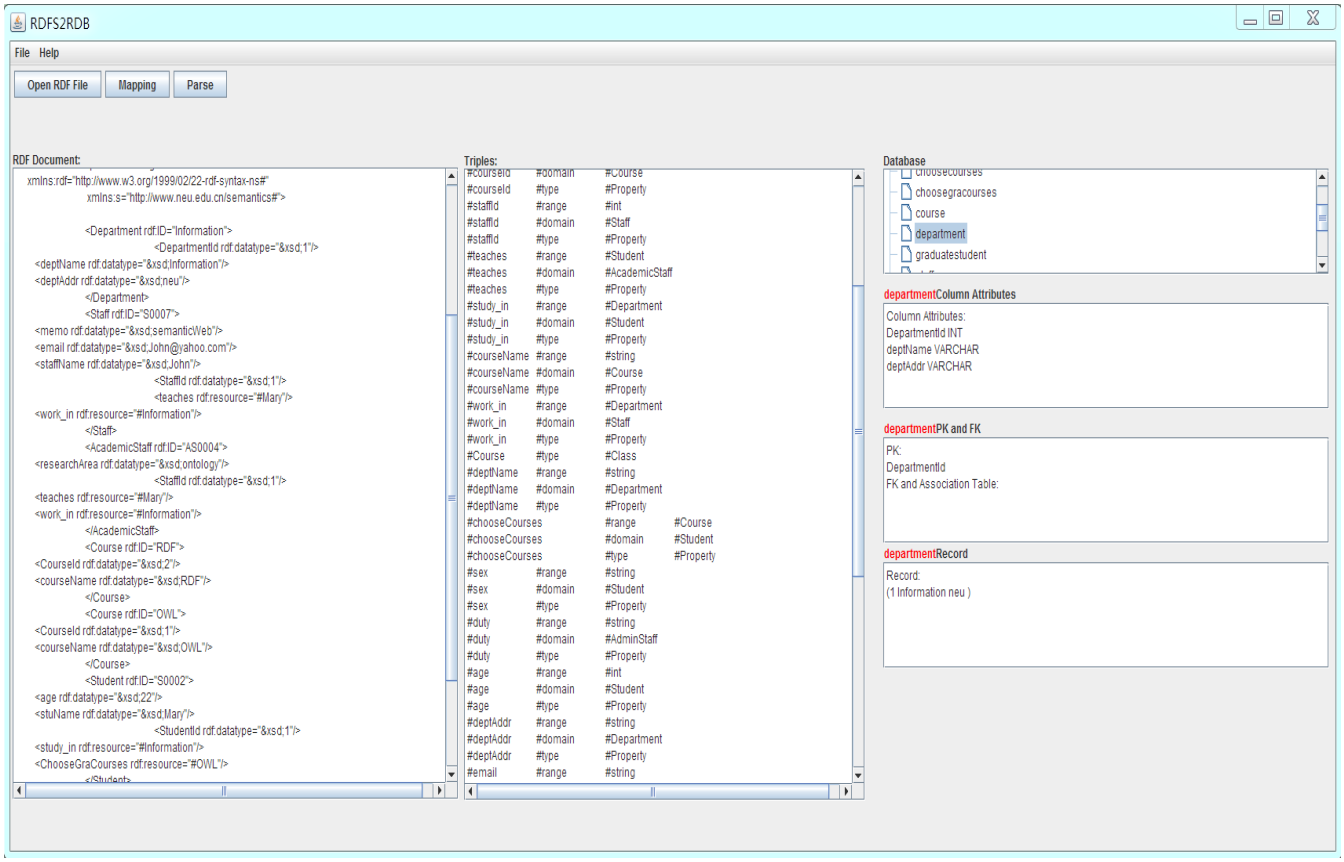


Figure 3. Screen snapshot of RDFS2RDB.

4.2. Experiments

In addition, in order to further verify the storage approach is information capacity preserving and correct storage and compare with the existing work, we also carried out some storage experiments of RDF(S) using the implemented tool RDFS2RDB, the data sets used in the experiments are mainly from the RDF(S) standard test data set LUBM (Lehigh University Benchmark) [13], and some ones (e.g., RDF(S) in Figure 2) are created manually by us with the RDF editor Protégé [19]. The experiments mainly cover the following two parts:

Firstly, we design some queries to query the original RDF(S) documents and the stored relational databases, where SPARQL is a W3C recommendation query standard for RDF(S), and SQL is the query standard for relational databases. Table 9 and Table 10 show several query statements (Q'_1-Q'_5), which are used to query the original RDF(S) documents in Figure 2 and the stored relational databases in Section 3.2, respectively. Here, Q'_1 queries the class hierarchies; Q'_2 queries the property hierarchies; Q'_3 queries the constraints of properties; Q'_4 queries the instances of classes; Q'_5 queries the relations of instances. The other queries can be done similarly. The results show that several query statements get the same results.

Secondly, we compare our work with the common and typical existing works (e.g., Horizontal, Generic/Vertical, and Specific/Binary storage patterns) as mentioned in section 1. After storing several different scale RDF(S) data sets (i.e., the numbers of RDF triples) in relational databases, we carried out some queries to the stored relational databases. Figure 4 shows the execution time of queries Q_1-Q_3. Here, Q_1 queries all of properties of a class; Q_2 queries all of instances of a class; Q_3 queries the property value of an instance; and the other queries are done similarly. In the experiments, each query is tested 10 times, and the average time is calculated as shown in Figure 4.

Table 9. The SPARQL query examples for the original RDF(S) document.

Name	SPARQL Query Condition	Results
Q'_1	SELECT ?*x* WHERE {?*x* rdfs:subClassOf dc:Staff.}	AdminStaff AcademicStaff
Q'_2	SELECT ?*x* WHERE {?*x* rdfs:subPropertyOf dc:chooseCourse.}	chooseGraCourse
Q'_3	SELECT ?*x*,?*y* WHERE {s:study_in rdfs:domain ?*x*. s:study_in rdfs:range ?*y*.}	Student Department
Q'_4	SELECT ?*x* WHERE {?*x* rdf:type dc:AcademicStaff.}	John
Q'_5	SELECT ?*x*,?*y* WHERE{{?*x* dc:Advices ?y.}	John Mary

Table 10. The SQL query examples for the stored relational database from the original RDF(S).

Name	SQL Query Condition	Results
Q'_1	SELECT Resource_Table.Localname FROM Resource_Table WHERE ID IN (SELECT SubCID FROM Resource_Table, SubClass_Table WHERE Resource_Table.Localname = 'Staff' AND Resource_Table.ID = SubClass_Table.SupCID)	AdminStaff AcademicStaff
Q'_2	SELECT Resource_Table.Localname FROM Resource_Table WHERE ID IN (SELECT SubProID FROM Resource_Table, SubPro_Table WHERE Resource_Table.Localname = 'chooseCourse' AND Resource_Table.ID = SubPro_Table.SupProID)	chooseGraCourse
Q'_3	SELECT Resource_Table.Localname FROM Resource_Table WHERE ID IN (SELECT Domain, Range FROM Resource_Table, Property_Field_Table WHERE Resource_Table.Localname = 'study_in' AND Resource_Table.ID = Property_Field_Table.ProID)	Student Department
Q'_4	SELECT Resource_Table.Localname FROM Resource_Table, AcademicStaff_Table WHERE Resource_Table.ID = AcademicStaff_Table.InsID	John
Q'_5	SELECT Resource_Table.Localname FROM Resource_Table, Relation_Advices_Table WHERE Resource_Table.ID = Relation_Advices_Table.SID OR Resource_Table.ID = Relation_Advices_Table.OID	John Mary

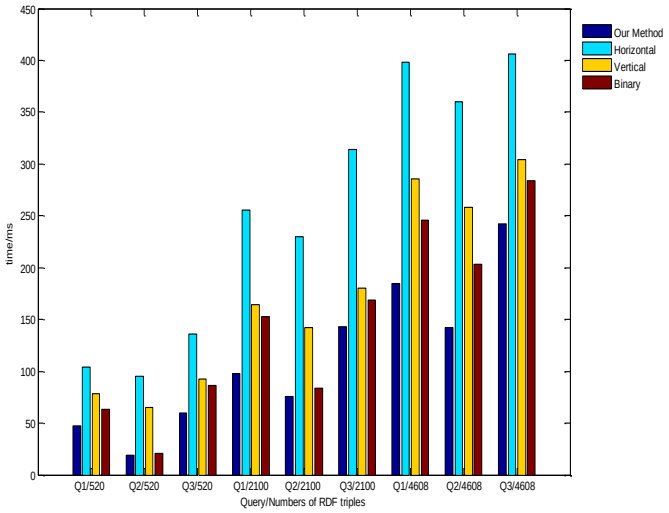


Figure 4. Comparison of query time among different storage models.

4.3. Discussions

Based on the observations above, the approach and tool in our work can store RDF(S) in relational databases. Moreover, comparing with the common existing work, it can be found that:

- Comparing with the Horizontal storage pattern as mentioned in section 1, when using the Horizontal storage pattern, only a relational table is created and its columns store all RDF(S) properties, and each RDF(S) individual instance is a record in the table. But in some RDF(S) data sets, different individual instance may contain different properties. Therefore, in this pattern, the created table may have many null values. In addition, each query needs to search all of the columns and tuples, and thus the query time may be increased.
- Comparing with the Generic/Vertical storage

pattern, when using such pattern, only a relational table is created, the table contains only three columns which are used to store the subject, predicate, and object of an RDF(S) triple. In this pattern, the semantic information of RDF(S) resources cannot be directly represented and stored, and each query needs to search all of the tuples in the table and execute the self-join operation.

- Comparing with the Specific/Binary storage pattern, when using such pattern, many tables may be created, and each table corresponds to an RDF(S) class or property. Each class table contains only one column which is used to store the RDF(S) individual instances belonging to the class. Each property table contains two columns which are used to store subjects and objects of the property. In this pattern, there are the potential scalability problems when the number of properties in an RDF(s) data set is high, since there may be many property tables in relational databases.

Of course, it should be noted that, as mentioned in section 1, RDF(S) information in some real applications is different in respects of scales and characteristics, and thus it is also difficult for us to give a unified pattern which is enough to effectively store all RDF(S) information. For example, if each individual instance has the same properties in some RDF(S) data sets, both of our approach and the existing Horizontal storage approach may be suited to store the RDF(S). In our work, we consider the semantic characteristics of RDF data and RDF Schema, the approach in our work creates the different relational tables for storing the different RDF(S) resources, and the tool and experiments show that the approach is feasible.

5. Related Work

With the development of RDF(S), lots of RDF(S) data sets have been created and they tend to become very large to huge. Therefore, one problem is considered that has arisen from practical needs: namely, efficient storage of RDF(S). The existing RDF(S) storage methods may be classified into several main categories: based on the file system, the special storage tools, and the databases. Among the storage methods above, according to their focuses, the storage of RDF(S) based on databases are closely related to our work. Therefore, in the following we will focus on the existing RDF(S) storage methods with databases.

In current, relating RDF(S) with databases becomes a topical problem since databases have the support of relatively mature theories and technologies. In general, the existing RDF(S) storage works mainly use the Horizontal, Generic/Vertical, and Specific/Binary patterns:

- Horizontal storage pattern: The work in [2, 5], investigated some related points of the Horizontal storage pattern of RDF(S), where only a generic table in the database is created, and its columns store all RDF(S) properties, and each RDF(S) individual instance is a record in the table. Also, some query optimization techniques (e.g., Hash) are introduced in the work.
- Generic\Vertical storage pattern: The work in [6, 10, 15, 23], investigated some related points of the Generic\Vertical storage pattern of RDF(S), where only a relational table is created, the table contains only three columns which are used to store the subject, predicate, and object of an RDF(S) triple.
- Specific\Binary storage pattern: The work in [3, 4, 18], proposed the Specific\Binary storage pattern. They proposed two types of property tables. The first type, which they call a clustered property table, contains clusters of properties that tend to be defined together. The second type of property table, termed a property-class table, exploits the type property of subjects to cluster similar sets of subjects together in the same table. Unlike the first type of property table, a property may exist in multiple property-class tables.

It should be noted that we do not cover all publications in the research area. The other kinds of methods and the comprehensive introduction of storage and reverse engineering can be found at [7, 9, 14, 17], in detail. In this paper, after we consider RDF instance data and the semantic characteristics of RDF Schema (e.g., the class hierarchies, the property hierarchies, the constraints of properties, multi-valued properties, the relationships between classes and properties, the namespaces, and etc.,), an RDF(S) storage framework and tool based on relational databases is developed, and also some related details are provided as shown in Sections 3 and 4 of this paper.

6. Conclusions

In this paper we investigated the storage of RDF and RDF Schema (collectively called RDF(S)) in relational databases, and proposed a formal approach and developed a prototype tool for storing RDF(S) in relational databases. By analysing the characteristics of RDF(S) data and schema semantic information in depth, an overall storage framework was developed first. On this basis, some detailed storage rules, a storage algorithm, and a storage example were given. Also, the correctness of the storage approach was discussed and proved. Based on the proposed storage approach, a prototype storage tool was implemented, and experiments and comparisons showed that the approach and the tool are feasible.

As mentioned in the existing work, it is difficult to give a unified pattern which is enough to effectively store all RDF(S) information. In our future work, we will further investigate the storage approach in depth,

test and compare with more existing storage models to improve the storage and query efficiency. Also some optimization techniques (e.g., Hash and Index) may be introduced. In addition, extending a database system with reasoning capabilities for supporting the reasoning of RDF(S) stored in databases is an important direction.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (61672139), the Natural Science Foundation of Liaoning Province, China (2015020048), and the Fundamental Research Funds for the Central Universities (N151704001).

References

[1] Aluc G., Ozsu M., and Daudjee K., "Workload Matters: Why RDF Databases Need a New Design," *in Proceedings of VLDB Endowment*, Hangzhou, pp. 837-840, 2014.

[2] Agrawal R., Somani A., and Xu Y., "Storage and Querying of E-Commerce Data," *in Proceedings of 27th International Conference on Very Large Data Bases*, Rome, pp. 149-158, 2001.

[3] Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., and Tolle K., "On Storing Voluminous RDF Description: the Case of Web Portal Catalogs," *in Proceedings of 4th International Workshop on the Web and Databases*, California, pp. 43-48, 2001.

[4] Abadi D., Marcus A., Madden S., and Hollenbach K., "Scalable Semantic Web Data Management Using Vertical Partitioning," *in Proceedings of 33rd International Conference on Very Large Data Bases*, Vienna, pp. 411-422, 2007.

[5] Bornea M., Dolby J., Kementsietsidis A., Srinivas K., Dantressangle P., Udrea O., and Bhattacharjee B., "Building an Efficient RDF Store over a Relational Database," *in Proceedings of ACM SIGMOD Conference on Management of Data*, New York, pp. 121-132, 2013.

[6] Broekstra J., Kampman A., and Van Harmelen F., "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," *in Proceedings of 1st International Semantic Web Conference*, Sardinia, pp. 54-68, 2002.

[7] Benslimane S., Malki M., and Bouchiha D., "Deriving Conceptual Schema from Domain Ontology: a Web Application Reverse Engineering Approach," *The International Arab Journal of Information Technology*, vol. 7, no. 2, pp. 167-176, 2010.

[8] Codd E., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, 1970.

[9] Faye D., Cure O., and Blin G., "A Survey of RDF Storage Approaches," *ARIMA Journal*, vol. 15, pp. 11-35, 2012.

[10] Harris S. and Gibbins N., "3Store: Efficient Bulk RDF Storage," *in Proceedings of 1st International Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, pp. 1-15, 2003.

[11] Jena, https://jena.apache.org/, Last Visited, 2016.

[12] Kaoudi Z. and Manolescu I., "RDF in the Clouds: a Survey," *VLDB Journal*, vol. 24, no. 1, pp. 67-91, 2014.

[13] Lehigh University Benchmark (LUBM). http://swat.cse.lehigh.edu/projects/lubm/index.htm, Last Visited, 2016.

[14] Modoni G., Sacco M., and Terkaj W., "A Survey of RDF Store Solutions," *in Proceedings of International Conference on Engineering, Technology and Innovation*, Bergamo, pp. 1-7, 2014.

[15] Ma L., Su Z., Pan Y., Zhang L., and Liu T., "RStar: an RDF Storage and Query System for Enterprise Resource Management," *in Proceedings of 13th ACM Conference on Information and Knowledge Management*, Washington, pp. 484-491, 2004.

[16] Miller R., and Ioannidis Y., "The Use of Information Capacity in Schema Integration and Translation," *in Proceedings of VLDB Endowment*, Dublin, pp. 120-133, 1993.

[17] Nitta K. and Savnik I., "Survey of RDF Storage Managers," *in Proceedings of 6th International Conference on Advances in Databases, Knowledge, and Data Applications*, Chamonix, pp. 1-6, 2014.

[18] Pan Z. and Heflin J., "DLDB: Extending Relational Database to Support Semantic Web Queries," *in Proceedings of 1st International Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, pp. 43-48, 2003.

[19] Protégé, http://protege.stanford.edu, Last Visited, 2016.

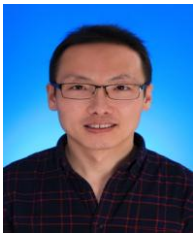
[20] RDF 1.1 Primer, W3C Working Group, 25 Feb 2014, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/, Last Visited, 2016.

[21] SPARQL 1.1 Overview, W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-overview/, Last Visited, 2013.

[22] van Rijsbergen C., "A New Theoretical Framework for Information Retrieval," *in Proceedings of 9th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Pisa, pp. 194-200, 1986.

[23] Wood D., Gearon P., and Adams T., "Kowari: a Platform for Semantic Web Storage and Analysis," *in Proceedings of 14th International Conference on World Wide Web*, Chiba, pp. 1-7, 2005.

[24] XML Schema Part 2: Datatypes Second Edition, http://www.w3.org/TR/xmlschema-2/, Last Visited, 2016.



Fu Zhang received his PhD degree in 2011 from Northeastern University, China. He is currently an associate professor in School of Computer Science and Engineering at Northeastern University, China. He has authored more than 40 refereed international journals and conference papers. His research work is published in high quality international conferences (e.g., CIKM and DEXA) and in highly cited international journals (e.g., Fuzzy Sets and Systems, Knowledge-Based Systems, and Integrated Computer-Aided Engineering). He has also authored two monographs published by Springer. His current research interests include knowledge graph, the Semantic Web, and knowledge representation and reasoning.

Qiang Tong received his PhD degree from Northeastern University, China. He is currently working in School of Software at Northeastern University, China. His research interests include RDF data management.

Jingwei Cheng received his PhD degree in 2011 from Northeastern University, China. He is currently working in School of Computer Science and Engineering at Northeastern University, China. He has authored more than 20 refereed international journals and conference papers. His current research interests include knowledge graph, the Semantic Web, and Description Logics.