ACSIJ Advances in Computer Science: an International Journal, Vol. 4, Issue 6, No.18 , November 2015
ISSN : 2322-5157
www.ACSIJ.org

A Method for Optimizing Maintenance and Querying Ontology-based Linked Data

Naghmeh Sohrabian[1], Bita Shadgar[2]

[1] **Department of Computer Engineering, Faculty of Engineering, Shahid Chamran University, Ahvaz, Iran**
sohrabian@isc.gov.ir

[2] **Department of Computer Engineering, Faculty of Engineering, Shahid Chamran University, Ahvaz, Iran**
bita.shadgar@scu.ac.ir

Abstract

At present, emerged technologies such as Resource Description Framework (RDF) are used to describe information in the semantic web. RDF triples are the basic components of linked data, which build the whole structure of the semantic web. Alongside the semantic web development, RDF data are also growing in scope and volume rapidly. As a result, the size of T-Boxes and also A-Boxes in linked data-related ontologies is undergoing a great change. The scale of ontology-based linked data requires efficient structures for storing and also querying on these data. This paper proposes a method based on relational databases for storing ontology-based linked data. This method achieves shorter query response time and more accuracy comparing other known RDF storage methods such as schema-oblivious, schema-aware and hybrid methods. To evaluate the results, DBpedia infobox ontology and dataset has been used.

Keywords: Linked Data, Ontology, Relational Database, Resource Description Framework, Indexing.

1. Introduction

Linked data come from different domains in various data sources on the semantic web. RDF links interlink these data and therefore in a near future make the whole semantic web connected. RDF triples are the basic components of linked data. They consist of three parts: subject, predicate and object. Subjects and predicates are identified with a unique global identifier named URI and object values can be URIs or literals. In recent years, linked data have grown so much in scope and volume [1]. DBpedia data source which converts Wikipedia data to the suitable format for the semantic web, is the nucleus for the semantic web [2] and it alone consists of billions of linked data available in about 100 languages. These data need efficient structures for maintenance and retrieval in a way that query response time and storage space size be acceptable.

This paper introduces a method for mapping ontology basic components to relational database components. It uses relational database for both linked data and ontology storage. It is desirable to store large ontologies with related instances in relational databases. Because Relational databases have long been used as primary sources for semantic web data and also ontology storage. They have also ensured the best facilities for storing, updating and querying the data from different domains [3-6] and they reduce the barriers for data exchange and integration. Furthermore using Relational Databases permits web application to query via SQL (Structured Query Language) instead of SPARQL (Simple Protocol and RDF Query Language), the semantic web query language which is not as matured as SQL in supporting the operations needed for querying data. SQL is relationally complete [5-7]; this means that any relational algebra operation such as select, projection, join and union can be modeled with SQL. SQL provides query capabilities using Data Manipulation Language (DML) and schema definition capabilities using Data Definition Language (DDL) [8].

Mapping ontologies to relational databases consists of three steps: schema mapping, data mapping and query mapping. Schema mapping builds the relational database schema based on the source ontology T-Box; data mapping converts RDF data to the relational tuples and query mapping translates SPARQL queries to SQL [9,10].

The rest of this paper is structured as follows. Section 2 shortly introduces existing methods for storing RDF data in relational databases along with their strong and weak points. Section 3 describes the proposed method. Section 4 compares the proposed method query response time with the other methods for different test queries and finally, conclusions and future works are discussed.

2. Related methods

Currently, there are several methods for mapping between RDF data model and relational databases [10,11]; however, all of them have some drawbacks, or are intended for certain purposes. These methods fall into four groups: (1) schema-oblivious method, (2) schema-aware method, (3) data-driven method and (4) hybrid method.

2.1 Schema-oblivious (also called generic or vertical)

One ternary relation (table) is used to store RDF triples. This table contains triples of the form <subject-predicate-object>. Fig. 1 shows the structure of the table. Different properties of a specific resource are tied together using the same subject URI. Attribute “subject” represents a resource that is the source of property, the property name is given in attribute “predicate” and attribute “object” represents a destination resource or literal value for the property. Well-known Schema-oblivious RDF stores include Jena [12,13], Sesame[14], KAON [15], RStar [16] and OpenLink Virtuoso [17].

triples

subject (resource URI)	**predicate (property name)**	**object (property value)**

Figure 1: Schema-oblivious storage method

2.2 Schema-aware (also called specific or binary)

This approach usually employs ontology to generate equivalent property relations and class relations in relational databases. Unlike the previous representation, one table per RDF/S schema property or class is used. A property table, Property(s,o), is created corresponding to each property in ontology and then stores each subject s and object o which are related by this property. A class relation, Class(i), is created for each class in ontology and stores instances i of this class. Fig. 2 shows the schema of the relational database. Representatives of schema-aware RDF stores are Jena [12,13], DLDB [18], RDFSuite [11], DBOWL [19], and PARKA [20]. This method considers a datatype proportionate to the type of datatype property in the related ontology.

Property$_1$

subject (resource URI)	**object (property value)**

class$_1$

subject (resource URI)

…

Property$_n$

subject (resource URI)	**object (property value)**

class$_m$

subject (resource URI)

Figure 2: Schema-aware method

2.3 Data-driven

This method uses RDF data instead of RDF schema or ontology, to generate database schema. For instance, database schema can be generated based on the patterns found in RDF data using data mining techniques. Property relations are created when their instances are first seen in an RDF document during data mapping. This method is seldom implemented in storage systems. It is used by sesame [14].

2.4 Hybrid

This method uses the combination of the features of the schema-oblivious and schema-aware methods. In this method, a schema-oblivious database representation is partitioned into multiple relations based on the data type of object o. So, property/class instances with range values of the same type are stored in the same relation and a binary relation, Class(i, c), is introduced to store instances i of classes c. Fig. 3 displays the relational database schema for this method.

Properties with range resource

subject (resource URI)	**predicate (property name)**	**object (property value)**

Class instances

subject (resource URI)	**object (classid)**

Properties with range integer

subject (resource URI)	**predicate (property name)**	**object (property value)**

Figure 3: Hybrid storage method

3. Method Description

The proposed method builds a storage system in which most kinds of related queries are answered in a relatively short time and reasonable storage space with more accuracy comparing the previous methods. Linked data and the ontology related to them are the inputs of the proposed system and relational database schema with ontology instances that are stored in relations are the output. This system uses DBpedia dataset infobox data and ontology[1] in order to test the proposed model. Fig. 4 shows the general structure of the proposed system. As Fig. 4 shows, the method consists of three main steps. The first and the primary one is transformation of ontology T-Box to relational database schema. In the second step, relational database schema is constructed based on DDL commands which have been generated in previous step. In the third step, relational tables are filled with linked data extracted from the dataset. These data are available in N-triples format.

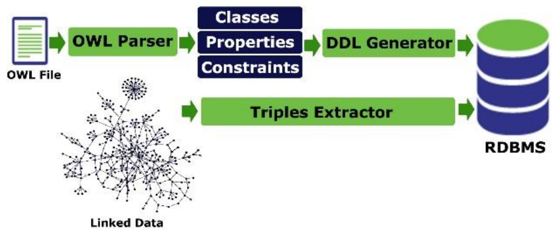

Figure 4: General structure of the proposed system

[1] Available for download at http://wiki.dbpedia.org/data-set-37

3.1 Translation of ontology T-Box to relational database schema

After validating the syntax of this file, ontology file is decomposed to its structural components such as classes, object properties, datatype properties and constraints. For each of these components separate DDL commands are generated in order to build the equivalent structure in relational database schema.

3.1.1 Transformation of ontology classes to relational structures

Fig. 5 shows the general steps for extraction of classes from the source ontology and generation of DDL commands for building proper relational structures. breath-first search is applied to the ontology file initially. Owl ontology class definitions are recognized with <owl:class> elements. First of all, owl:thing class, which is the parent of all ontology classes is added to a queue. After that, in each hierarchy level, classes are observed one after another and their attributes take proper values. For any class definition in ontology file, the class attributes such as rdf:about, rdfs:label, rdfs:subclassof and rdfs:comment are given proper values. DDL command corresponding to create a relational database is the first command to be written in the output file. After that, another DDL command is written for generating the meta table named classes with the schema seen in Fig. 6. This meta table stores the general information for any class in ontology. Attribute state stores “non-leaf” in case of root classes and “leaf” in case of “leaf” classes. When class definition search in ontology ends, DDL commands to fill table classes with proper data are written.

3.1.2 transformation of object/datatype properties to relational structures

This step is somewhat similar to the previous one. Here object and datetype properties definitions in ontology file are used to fill meta table named Property. They are recognized with <owl:objectproperty> and <owl:datatypeproperty> elements respectively. The first DDL command is written to generate meta table properties in relational database schema. This table stores the meta data for ontology properties. Fig. 7 shows the schema for this table. Domain and range attributes store URIs of the source and target of each property. As OWL does not contain any data type itself, it uses data types from XML schema. Attribute flag is used to distinguish between object and datatype properties. In the proposed method, MySQL 5.5 is used as RDBMS. Therefore, data types in OWL ontology file should be mapped to proper data types in MySQL. Similar to hybrid method, property tables are categorized based on object’s data type. Fig. 8 displays their schema.

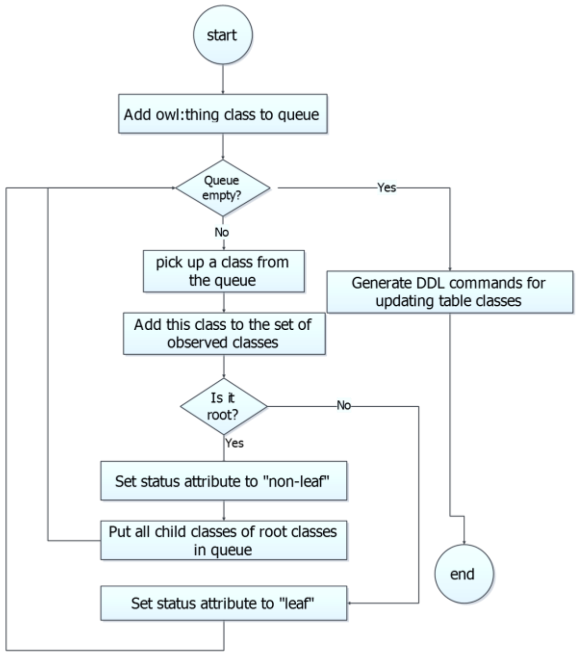
start
Add owl:thing class to queue
Queue empty?
Yes
No
pick up a class from the queue
Generate DDL commands for updating table classes
Add this class to the set of observed classes
Is it root?
No
Yes
Set status attribute to "non-leaf"
Put all child classes of root classes in queue
end
Set status attribute to "leaf"

Figure 5: Transformation of ontology classes to relational structures

classID	title	URI	state	comment

Figure 6: the schema of table classes

propertyID	title	URI	domain	range	flag

Figure 7: The schema of table properties

statementID	subject	predicate	object

Figure 8: The schema of property tables for storing instances

These relations are the most important ones. They are used to store triples whose predicates are datatype properties and types_resources table is used to store triples whose predicates are object properties. For each triple the property(predicate) range specifies where to store the triple.

3.1.3 Application of ontology relations to database schema

In this step, all the relations in the source ontology are transformed to relational structures. Based on the input ontology there exists various kinds of relations. Storing these relations can be useful while inferencing new data from existing RDF triples. In DBpedia ontology, relations come in four groups: rdfs:subclassof, owl:equivalentclass, owl:equivalentproperty, owl:functionalproperties. For subclasses, a table with two columns is generated: one for class URI and another for parent class URI. In order to store equivalent classes and equivalent properties two tables with two columns are generated which store class/property URI and equivalent class/property URI. Functional properties are stored in a single column table. One of the strength points for the proposed method is the generation of a meta table named propertyClass_instances. As Fig. 9 shows, for each property this table stores a unique identifier named ID, property URI, classID which points to a class in table classes and flag which distinguishes between object and datatype properties. Another attribute named table_ is the name of the table which is going to contain RDF statements of a specific property in the next step. Therefore, there is no need to include all property tables in queries. Instead, only the retrieved tables are used for querying in proposed approach. Using this table facilitates and accelerates the queries which ask for the instances of a particular property. Querying the individuals of a particular class is the same story. Again, the attribute table_ value is used as a reference for the storage table containing RDF triples. So, this table also facilitates the queries on all RDF statements that are related to a specific class.

ID	property	classID	flag	table_

Figure 9: The schema of propertyclass_table

The proposed method generates another table named resourceClass which links each resource to the class which it belongs to. This table has two columns: one stores resource URIs and another one stores class URIs. This table facilitates and accelerates the queries which ask for the parent classes of each resource.

3.2 Generation of relational database

Before loading linked data into relational tables, the whole schema of relational database should be built. Executing DDL commands which are generated in previous steps builds the ontology T-Box. Moreover, meta tables classes and properties are filled with proper RDF data.

3.3 Filling relational database with extracted linked data

The entry to this step is the relational database schema that has been generated in previous step. Here the RDF triples are converted to the relational tuples depending on the target table. In DBpedia dataset, linked data are stored in infobox properties and infobox specific properties parts. Objects are stored as simple or typed literals. Therefore, they include extra texts such as language labels, XML data type URIs and some extra characters such as "", "<", ">" and so on. To extract related RDF triples out of this dataset, objects of the datatype properties should be modified in a way that these extra texts are removed and the genuine object is retrieved. To find the proper table for storing each RDF triple, the value for attribute table_ is used.

4. Results Evaluation

In this section, after application of all these methods to DBpedia dataset, the query response time and storage space are compared with the proposed method (with or without indexing) in case of queries with different viewpoints: queries on linked data structural components such as subject, predicate, object, queries, queries on resources' parent classes and queries on class-related linked data. Then, the storage space of the proposed method is compared with the other methods.

4.1 Capability of response to different query types

4.1.1 Queries which ask for linked data subjects

In schema-oblivious method, the below SQL command retrieves RDF statements having a particular subject identified with the subject URI:

```
select subject,object,predicate from triples where
subject=@subjectURI
```

In schema-aware method, all property tables should be searched which is very slow and inefficient. Because for each property table, a union operation is added to the SQL query. The SQL query generated is as follows:

```
select subject,object,predicate from property1 where
subject=@subjectURI
union
select subject,object,predicate from property2 where
subject=@subjectURI
union
.
.
union
select subject,object,predicate from propertyn where
subject=@subjectURI
```

property$_1$ to property$_n$ are the first and last property tables which contain RDF triples respectively. In hybrid method, always the same number of tables is explored. This number depends on the number of data types which are defined in ontology. In case of DBpedia infobox ontology this number equals 11. The SQL command is as follows:

```
select subject,object,predicate from types_1 where
subject=@subjectURI
```

union
.
.
union

select subject,object,predicate **from** types_11 **where** subject=@subjectURI

The proposed approach first retrieves the property table names that contain the specified subject via an SQL query as follows:

(1) **select** **distinct** table_ **from** propertyClass_instances, propertyClass_table **where** propertyClass_instances.classID= propertyClass_table.classID **and** resource=@resourceURI

Then, a union operation is applied to the retrieved tables:

(2) **select** subject,object,predicate **from** types_1 **where** subject=@subjectURI
union
.
.
union
select subject,object,predicate **from** types_n **where** subject=@subjectURI

As the number of queried tables decreases, the query speed increases in comparison with the previous method. In order to increase this speed even more, an index is added to subject column in all property tables. Query speeds in each case are evaluated with three random URIs in DBpedia dataset. Fig. 10 shows the results for this kind of query in terms of time. The parameter for the query1 is the first URI, for the query2 is the second URI and for query3 the third one. As seen in Fig. 10, the proposed method response time is decreased about 47 percent in case of query1, 44 percent in case of query2 and 40 percent in case of query3. When indexing is applied to column subject, response time decreases 35 percent comparing the situation without indexing in case of query 1, 59 percent in case of query 2 and 60 percent in case of query 3.

4.1.2 Queries which ask for linked data predicates

In schema-oblivious method, the below SQL command retrieves linked data predicates.

select subject,object,predicate **from** triples **where** predicate=@propertyURI

In schema-aware method, a simple SQL query retrieves the specified data:

Select subject,predicate,object **from** property[x]

property[x] is the property table which contain the specified data. This method is very efficient in response to queries of this type.

Hybrid method behaves the same as querying on subjects, but instead it asks for predicates.

The proposed approach retrieves the names of property tables which contain the specified predicate:

select **distinct** table_ **from** propertyClass_table **where** property=@propertyURI

Then, a union operation is applied to retrieved tables:

select subject,object,predicate **from** types_1 **where** predicate=@propertyURI

union
.
.
union

select subject,object,predicate **from** types_n **where** predicate=@propertyURI

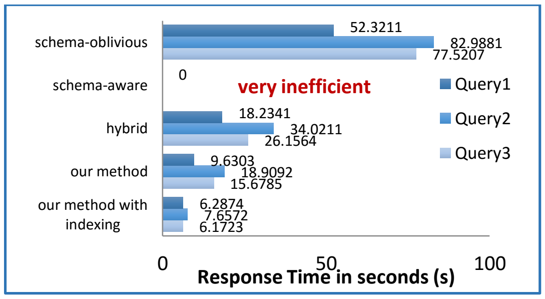

Figure 10: Comparison of response time for query on linked data subjects in seconds (s)

So, similar to the previous condition with decrease in number of queried tables, query speed increases. Additionally, applying index to predicate columns increases this speed even more. It should be notified that the proposed method infers new triples from the main ones and adds them to the retrieved results. For this purpose, the similar properties to the queried property are searched using table sameProperties. However, to avoid increasing the storage space this method does not store inferred triples in any structure. Instead, it adds them to the main triples in run time. Fig. 11 shows the results of query responses in this case.

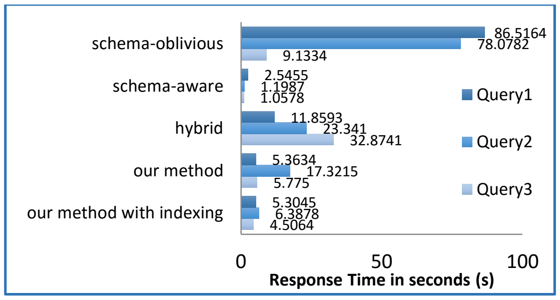

Figure 11: Comparison of response time for query on linked data predicates in seconds (s)

4.1.3 Queries which ask for linked data objects

This kind of query needs two parameters: property URI and specific range for property values. So, here just the data property case is studied. In schema-oblivious method, no SQL query can extract the object value out of the third part of RDF triple. In schema-aware method, a SQL query similar to the one on predicates in this method retrieves the objects in special range of values. Again, the method applies union operation to so many tables and therefore results in bad query results. Hybrid method behaves the same as querying on subjects and predicates, but instead it asks for predicates. The proposed approach here is similar to the previous one, but here the objects are queried in specific ranges. Fig. 12 shows the time results of all methods in case of queries on objects. In order to evaluate the time performance of queries, three random URIs with random ranges are selected. The results show that when there is no indexing, schema-aware method performs the best. But totally, the proposed method with indexing is the best in terms of time. It shows 43 percent decrease in time in case of query1, 8 percent in case of query2 and 82 percent in case of query3.

4.1.4 Queries on resource parent classes

As schema-oblivious method lacks the class information, this kind of query cannot be applied to this method. Both schema-aware and hybrid contain structures for storing the instances of a specific class, but lack the possibility of querying on the parent classes that are related to a resource. The proposed approach uses table resourceClass to ask for class instances.

select classURI **from** resourceClass **where** resource=@classURI

For each individual, the proposed method only stores leaf classes in ontology tree. After retrieving a particular class URI it refers to table subclasses to append all the parent classes for the specified resource to the list of retrieved classes. Furthermore, it queries table sameClasses to find the equivalent classes with the ones that are retrieved as resource parent classes and adds the results to the previous retrieved classes. Finally, the proposed method adds an index on column resource to increase the query speed. Fig. 13 shows the results of this query in terms of time for three random class URIs.

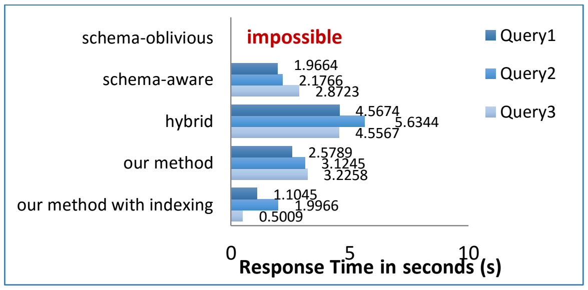

Figure 12: Comparison of response time for query on linked data objects in seconds (s)

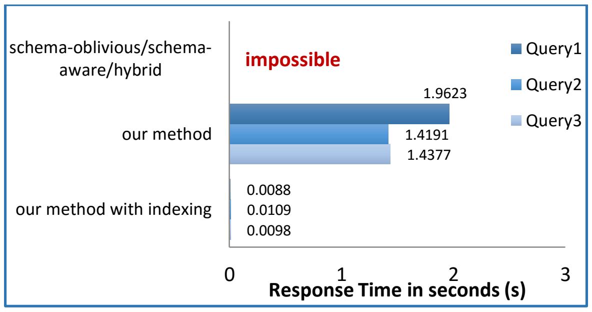

Figure 13: Comparison of response time for query on resource parent classes in seconds (s)

The results show that indexing decreases query response time about 99 percent in case of query1, 80 percent in case of query2 and 99 percent in case of query3.

4.1.5 Queries on class-related linked data

This kind of query retrieves all linked data that are related to a particular class in the form of RDF triples. As schema-oblivious method lacks the class information, this type of query is not possible to execute in this method. In schema-aware and hybrid method the SQL query which retrieves class-related linked data is as follows:

select subject,object,predicate **from** class[x],property$_1$ **where** class[x].resource=property$_1$.subject
union
.
.
union
select subject,object,predicate **from** class[x],property$_n$ **where** class[x].resource=property$_n$ .subject

n is the number of properties. As there are so many join and union operations, this method time performance is very inefficient. The proposed method uses table propertyClass_instances. First a SQL query on table propertyClass_instances retrieves all the property and target table names for storing the RDF statements that are related to a specified class:

select property,table_ **from** propertyClass_instances **where** class=@classURI
select subject,predicate,object **from** types_1 **where** predicate=@propertyURI
union
.
.
union
select subject,predicate,object **from** types_n **where** predicate=@propertyURI

So, the proposed method retrieves all the class-related linked data without joining any tables. This increases the query speed. Then, an index is added to column predicate which increases query speed even more. Fig. 14 shows the response time for this type of query in all methods.

4.2 The storage space

The storage space of different methods can be investigated here from two points of view: number of generated tables and the volume of relational database.

4.2.1 Number of generated tables

As the number of generated tables increases, the storage and retrieval overhead also increases. Furthermore, data management and updates get harder. The schema-oblivious method uses just one table for the storage of RDF statements. All extracted linked data are stored in this table. As previously mentioned, this method lacks any structure for the storage of ontology properties and classes. Application of this method to the DBpedia infobox dataset results in about 14,000,000 RDF triples being stored in one table. The schema-aware method considers a table for each class or property in ontology. Application of this method to dataset results in generation of 314 tables for classes, 851 tables for object properties and 893 tables for datatype properties. Hybrid method generates a table for each class to store the instances of that class. For each group of property data types, a table is generated to store the RDF triples. Application of this method to dataset results in generation of 314 tables for the storage of classes and 10 tables for the storage of datatype properties. The proposed method does not consider any structure for the storage of classes. It generates one table for the storage of object properties, 10 tables for datatype properties and 8 meta tables for storing general information. Table 1 represents the number of tables for each method.

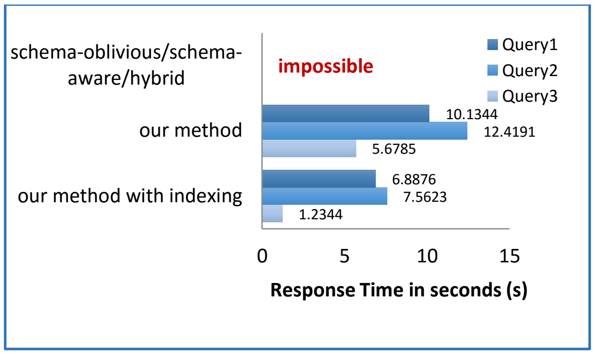

Figure 14: Comparison of response time for query on class-related linked data in seconds (s)

Table 1: The number of generated tables

Storage method	Number of relations
schema-oblivious	1
schema-aware	2058
hybrid	325
our method	**18**

4.2.2 The relational database volume

Obviously, as the volume of database increases, the storage and retrieval overhead also increase. Table 2 shows the total database volume for each method. It shows that the proposed method generated database is the lowest in volume.

Table 2: The total generated database volume in Gigabyte

Storage method	Database Volume
schema-oblivious	2.2 Gigabyte
schema-aware	3.92 Gigabyte
hybrid	2.45 Gigabyte
our method	**2.13 Gigabyte**
our method with indexing	2.23 Gigabyte

5. Conclusion

In previous section the response performance of queries on ontology-based linked data and the storage volume for existing methods such as schema-oblivious, schema-aware and hybrid methods and proposed method are investigated and compared. The results show that the schema-oblivious method is only efficient in response to queries which ask for subjects, predicates or objects of Linked data. This method lacks any structure for the storage of classes or properties. It stores all RDF triples in one table. This causes problems with query speeds when performing some operations like joining the table with itself. Schema-aware generates tables for any class or property in ontology. This causes an overhead on whole the database and makes data management and updates hard. Furthermore, for most of the queries so many union operations should be included. This method performs well just in response to the queries which ask for class individuals or instances of a particular property. Hybrid method performs well in response to queries which ask for specific range of values in addition to the query types which are supported by schema-aware method. Hybrid method has resolved the problem with the number of generated tables in schema-aware method, but it still contains all property tables in some queries. The proposed method aims at resolving the problems with the previously discussed methods. Furthermore, it can respond well to all the query types which are mentioned in this article. It uses indexing on queried data column to speed up the queries and uses inference to increase the accuracy of the retrieved RDF data. Furthermore, the number of generated tables is independent of the number of classes in ontology. The results show that in most of the cases, the proposed method with indexing performs the best in terms of response time, result completeness and simplicity of queries which are used to retrieve data and it supports most types of queries comparing the other methods.

References

[1] C. Bizer, F. Universitat, T. Heath, T. Berners-Lee, "Linked data - the story so far", *International Journal on Semantic Web and Information Systems*, Vol. 5, No. 3, pp. 1-22, 2009.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, "DBpedia: a nucleus for a web of open data", in *The 6th International Semantic Web Conference (ISWC2007)*, Busan, Korea, November 2007.

[3] M. d. M. Roldan-Garcia, J. F. Aldana-Montes, "A Survey on Disk Oriented Querying and Reasoning on the Semantic Web", *in The 22th IEEE ICDE Workshop SWDB,* Atlanta, 2006.

[4] D. Beckett, J. Grant, "SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes", (last modified: 23 January 2003), [accessed: 11 April 2012], <http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/>.

[5] D. E. Spanos, P. Stavrou, N. Mitrou, "Bringing Relational Databases into the Semantic Web: A Survey", *Semantic Web*, Vol. 0, No. 0, pp. 1-41, 2010.

[6] I. Astrova, N. Korda, A. Kalja, "Storing OWL Ontologies in SQL Relational Databases", *in proceedings of the World Academy of Science, Engineering and Technology,* 2007.

[7] D. E. Spanos, P. Stavrou, N. Mitrou, "Bringing Relational Databases into the Semantic Web: A Survey", *Semantic Web*, Vol. 0, No. 0, pp. 1-41, 2010.

[8] C. J. Date, *An Introduction to Database Systems*, 8th edition, Addison Wesley , Boston, 2003.

[9] W3C Incubator Group, "A survey of current approaches for mapping of Relational Databases to RDF", (last modified: 8 January 2009), [accessed: 5 April 2012], <http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf >.

[10] A. Chebotko, S. Lu, X. Fei, F. Fotouhi, "RDFPROV: A relational RDF Store for querying and managing scientific workflow provenance", *Data & knowledge Engineering*, Vol. 69, No. 1, pp. 836-865, 2010.

[11] Y. Theoharis, V. Christophides, G. Karvounarakis, "Benchmarking Database Representation of RDF/S Stores", *in Proceedings of the 4th International Semantic Web Conference(ISWC2005), LNCS 3729,* Galway, Ireland, 2005.

[12] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, "Efficient RDF Storage and retrieval in Jena2", *in the first International Workshop on Semantic Web and Databases*, Berlin, Germany, 2003.

[13] B. McBride, "Jena: Implementing the RDF Model and Syntax Specification", *in proceedings of the second international workshop on semantic web(semweb2001)*, Hong Kong, China, 2001.

[14] J. Broekstra, A. Kampman, F. V. Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema", *in Proceedings of the first International Semantic Web Conference(ISWC2002),* Chia, Sardinia, Italy, June 2002.

[15] T. Gabel, Y. Sure, J. Voelker, "KAON – An Overview: Karlsruhe Ontology Management Infrastructure", *University of Karlsruhe,* 2004.

[16] L. Ma, Z. Su, Y. Pan, L. Zhang, T. Liu, RStar: An RDF Storage and Query System for Enterprise Resource Management, *In proceedings of the International Conference on Information and Knowledge Management (CIKM),* Washington, DC, USA, 2004.

[17] O. Erling, Implementing a SPARQL compliant RDF triple store using a SQL-ORDBMS, Technical report, OpenLink Software Virtuoso, 2001, Available from http://virtuoso.openlinksw.com/wiki/main/Main/VOSRDFWP.

[18] Z. Pan, J. Heflin, DLDB: Extending Relational Databases to Support Semantic Web Queries, *In Proceedings of the International Workshop on Practical and Scalable Semantic Web Systems (PSSS),* Sanibel Island, Florida, USA, 2003.

[19] S. Narayanan, T. M. Kurc, and J. H. Saltz. DBOWL: towards extensional queries on a billion statements using relational databases. Technical Report OSUBMI_TR_2006_n03, Ohio State University, 2006. Available from http://bmi.osu.edu/resources/techreports/osubmi.tr.2006.n3.pdf.

[20] K. Stoffel, M.G. Taylor, J.A. Hendler, Efficient Management of Very Large Ontologies, *In proceedings of the American Association for Artificial Intelligence Conference(AAAI),* Palo Alto, California, 1997.