CONVERTING DB TO RDF WITH ADDITIONAL DEFINED RULES

Mamdouh Farouk and Mitsuru Ishizuka
Creative Informatics Department, The University of Tokyo, Tokyo 1138656, Japan

Keywords: DB to RDF, Inference Rules, Searching RDF.

Abstract: In linked data, web resources and data are represented into RDF. The new web of linked data should be completely machine-understandable. Moreover, since the start of Linking Open Data project, more and more providers publish linked data. Therefore, converting DB into RDF is important because a large amount of web data stored in databases. This work presents an approach for converting DB to RDF with additional inference rules. The generated data contains not only RDF data that represents relational DB but also additional discovered relation based on a set of predefined rules. Moreover, this paper proposes a simple search engine, which consumes the generated data and the defined inference rules. A prototype for the proposed approach and results of experiments show the effectiveness of the proposed approach.

1 INTRODUCTION

In Semantic web vision, web agents can understand web data and do actions to help the user (Berners-Lee , 2001). To enable web agent to understand web content, web data should be represented into a machine understandable format. Moreover, there are many languages formalized to represent web data and resources such as RDF, DAML and OWL. The Resource Description Framework (RDF) is a W3C recommendation that represents current web into machine understandable format.

Further, a huge amount of web data is stored in databases (Siegfried, 2003). Therefore, many researchers pay much care to convert relational DB to RDF triples. Moreover, many researchers try to represent dynamic web pages, which retrieve their content from underlying DB, into semantic format (Zhuoming, 2006)(Mamdouh, 2005). In the other hand, the process of converting DB to RDF should be simple (Svihla, 2005) to encourage the DB owner to convert his data.

There are different approaches to convert DB to RDF (Siegfried, 2003) (Pan, 2003) (Ismael, 2004). A common step in these approaches is mapping between DB schema and ontology structure. Based on this mapping, the DB can be accessed semantically either by generating RDF triples corresponding to original data or by keeping the data in the DB, where it can be managed better, and generating RDF on demand. There are different approaches for the latter way. One approach is converting SQL query result to RDF on the fly when the DB is queried (Svihla, 2005). This approach is suitable in case of dynamic web pages that retrieve content from underlying DB. Another approach is developing a semantic access layer as an intermediate layer between web agents and normal DB (Ismael, 2004).

Although, the main objective of converting relational DB to RDF is to enable web agents to understand this data, there are some difficulties facing web agents to understand this data. One important issue that should be faced is finding implicit data. In other words, how the web agent can infer the implicit data like a human who read the normal web pages. Showing this implicit data will enables web agent to deeply understand web data. For example, a query asks about an author who is interested in *semantic web* that is run over a corpus such as semantic web conference corpus may return no result. The answer of such query already exists in the RDF data but the query answering process cannot get the answer because the answer implicitly exist.

Although, DB is an excellent tool to store and manage data, it needs simple inference to improve its performance of querying data (Pan, 2003). This work is an extension to DB2RDF approach that converts DB to RDF data. This paper does not focus on converting DB to RDF. However, it focuses on

Farouk M. and Ishizuka M..
CONVERTING DB TO RDF WITH ADDITIONAL DEFINED RULES.
DOI: 10.5220/0003939907090716
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 709-716
ISBN: 978-989-8565-08-2

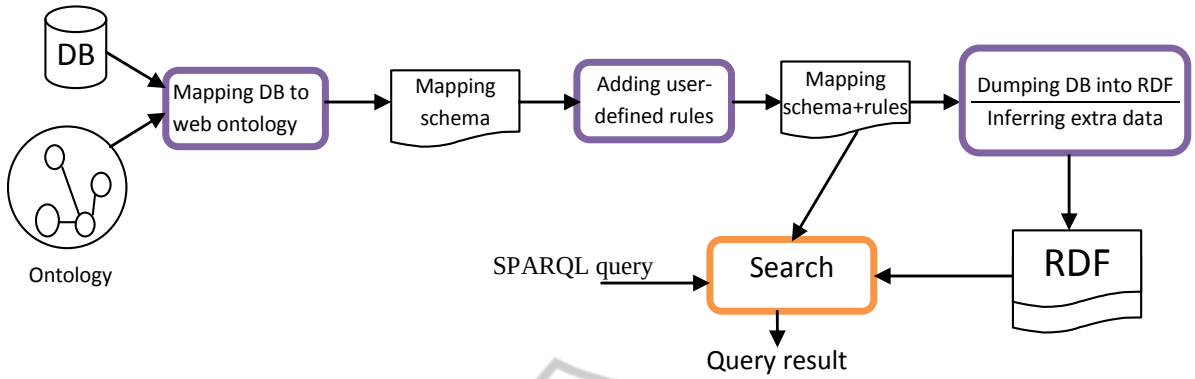

Figure 1: System architecture.

adding extra knowledge (user-defined rules) during mapping process. These rules are useful to discover extra relations. Using these rules, web agents can understand web data easily. In the proposed approach, the generated RDF data contains not only original DB data but also inferred data that supports query answering process.

This paper proposes an approach to convert relational DB to RDF with additional relations discovered based on user-defined rules. Unlike other approaches, our approach provides not only mapping and generating RDF but also adding extra knowledge, which is very useful in query answering process. In other words, this work proposes adding extra knowledge (user-defined rules) to the mapping schema level to improve the query-answering process. The generated RDF semantic representation together with the added knowledge can be used by intelligent search engines to infer more data and obtain accurate search results. Moreover, an extension to SPARQL search engine which exploits the added rules to answer more queries is presented.

One approach that maps and converts DB to RDF is D2R (Chris, 2010). D2R tool auto-generates the mapping file and the user should modify this generated file to fit the appropriate meaning. Moreover, D2R server enables the user to query the DB using SPARQL queries. Dumping RDF data that represents DB is also supported by D2R.

Moreover, a related approach, which tries to express rules and infer additional RDF data, is SPIN (Holger, 2011). SPIN is a group of RDF properties that can be used to express rules. These rules attached to a specific ontology class and can be applied to infer data, or modify the current data. *spin:rule* property can be used to defined an inference rule using SPARQL construct or insert/delete.

Moreover, SPIN adds rules to ontology level. However, our approach separates between rules level and ontology level. Separation between ontology and rules levels gives the user flexibility to add rules. In other words, it is difficult for the user to update the standard shared ontology to add his rules. Moreover, there are many users may add rules to infer the same property depending on their own data. The user wants to extend his data depending on the semantics of the data and the expected queries to be asked. Therefore, the users have different data want to make many rules even for the same ontology. Attaching rules to dataset gives flexibility to the users and avoids rules conflicts on ontology level.

The remainder of this paper is organized as follows. Section 2 describes the overall system architecture. Section 3 explains the proposed approach for converting DB to RDF. Section 4 describes an extension for SPARQL search engine that exploits the generated RDF and defined rules. The experiments and results are discussed in section 5. Finally, section 6 provides the conclusion of this research.

2 SYSTEM ARCHITECTURE

The proposed system is divided into two main parts. The first part is converting relational DB to RDF. The second part is searching the generated RDF triples using SPARQL queries.

As shown in figure 1, there are three tasks for representing DB into RDF: the first task is mapping between DB schema and ontology. The second process is adding user-defined rules to the generated semantic schema. The last task is generating RDF data. This task includes two sub-processes which are generating RDF data represents relational database and generating RDF data represents the inferred RDF triples using the defined rules. A simple search engine is designed to query the generated RDF using SPARQL queries. The search engine exploits the

predefined rules to improve query answers.

Moreover, two ways for using the user-defined rules are applied. Forward chaining is applied during the process of generating RDF to add the inferred RDF triples. However, backward changing is applied during the searching process to use the rules without storing the inferred data.

3 CONVERTING RELATIONAL DB TO RDF

In the proposed approach, the mapping between relational DB and RDF is a manual process in which the user maps between schema of DB and ontology structure. The user uses the developed tool, figure 2, to map between DB and different ontologies. The generated mapping is expressed into XML intermediate format.

3.1 Mapping DB to Ontology

There are three steps for this mapping process. The first step is mapping DB tables to ontology classes. In this step, the user selects ontology class corresponding to each table. The user can select classes belong to different ontologies. The second step is mapping between DB fields and ontology properties. The user selects a suitable property for each field. The mapping tool helps users to do this mapping easily and correctly. The last step is mapping relations of the DB. In this step, the user represents M-M and 1-M relation in terms of

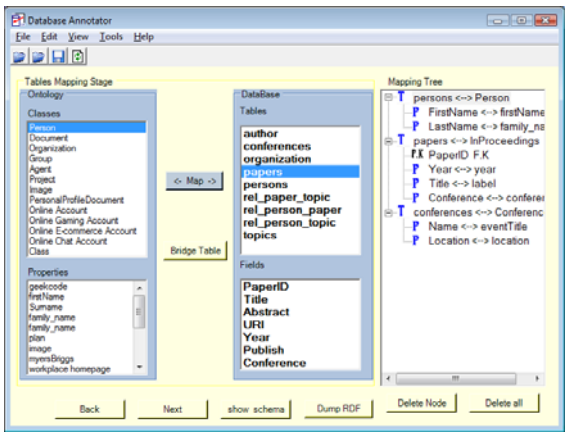

Figure 2: The mapping tool.

ontology relations. M-M relation is considered as two relations each one is 1-M relation. The user maps the foreign key field to the appropriate ontology property that represents same relation between ontology classes. For example, consider a DB for university researchers contains two tables: *researchers*, and *departments*. The field *deptID* in *researchers* table is a foreign key refers to *departments* table. In such case, the user may map *deptID* field to *hasaffiliation* property in *person* class. The domain of *hasaffiliation* property is *organization* class. *hasaffiliation* property represents the relation between *researchers* table and *departments* table.

3.2 Adding Rules

The next step after mapping between DB schema and ontology is adding extra knowledge to the mapping file. This extra knowledge is considered as an extension for the original data stored in the DB. Moreover, this knowledge is used to infer more data from the DB and to support query answering process.

To clarify the idea of adding user-defined rules, consider this scenario. The database of international semantic web conferences (ISWC) contains information about some conferences in semantic web field and other related data such as published papers, authors and so on. Figure 3 shows the schema of ISWC DB. Normally, this database is queried about authors and their interest points or their publications. For example, who is interested in "semantic representation"?. Who knows Prof. John? The DB or the traditionally generated RDF data cannot properly answer these questions based on the available data. Moreover, neither in DB nor in the generated RDF data contains *knows* relation in foaf ontology. However, a human can suggest an answer based on a simple inference. Consequently, adding some inference rules helps web agents to understand the data and answer such queries. For example, the following rules can be added:

- If a person A is an author to a paper Y, and a person B is an author to the same paper Y → A knows B.
- If a person A is an author to a paper Y, and the main topic of Y is T then → A is interested in T.

Using these rules is considered as a DB extension that adds more relations to the original relational database. For example, the above DB contains little information about research points of authors. However, the second rule enriches this information by discovering more research points for authors depending on their publications. As a result, these rules enable search engines to answer more queries. Adding user-defined rules depends on the meaning of the DB schema and the queries that used to be asked.

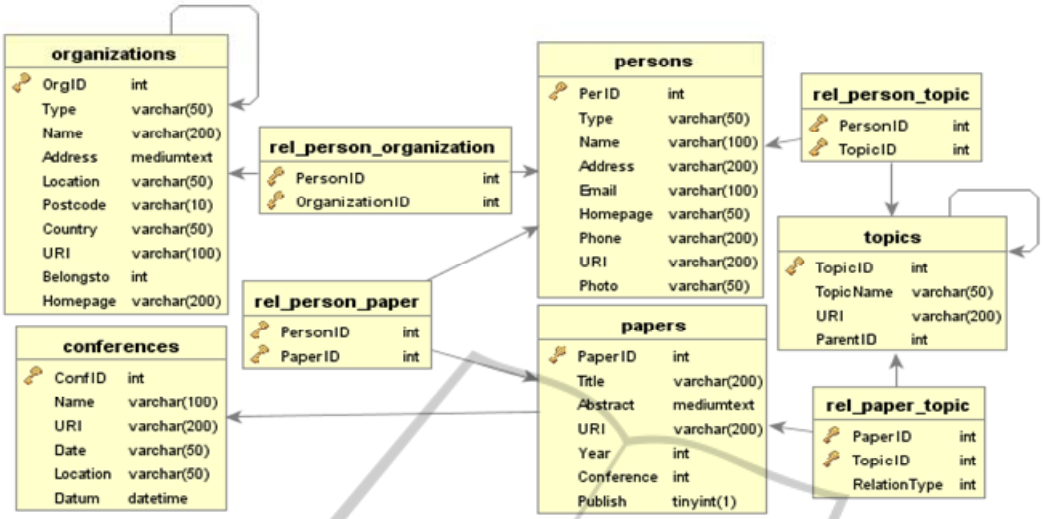

Figure 3: ISWC DB schema.


```
<rule id="2" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:iswc="http://annotation.semanticweb.org/iswc/iswc.d
aml#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
   <text>
    if two people are authors for the same paper, then they
know each others.
   </text>
<condition>
<con>
{
 ?ppr rdf:type iswc:InProceedings.
 ?person rdf:type foaf:Person.
 ?person2 rdf:type foaf:Person.
 ?ppr dc:Creator ?person.
 ?ppr dc:Creator ?person2.
 FILTER (?person != ?person2)
}
</con>
</condition>
<action>
{   ?person foaf:knows ?person2.  }
</action>
 </rule>
```

Figure 4: Example of user-defined rules.

These rules are added during the mapping process, which occurs only once. Using these rules solves the problem of finding the implicit information and enables web agents to go one more step to understand web data.

Production rule format, “condition → action”, is used to represent the user defined rules. The syntax of production rule is carefully designed to be easy for implementation of generating extra data and to be easy for reasoning. The condition part syntax is the same as SPARQL query condition syntax. The action part is also represented into SPARQL syntax to be easy for execution. Figure 4 shows an example for the added rules. The first part of the rule is xml namespaces for the used vocabularies. The second part is the conditions of the rule represented into SPARQL syntax. The last part is the action part, which is true if the conditions are true.

Moreover, there are two approaches to use these rules. The first one is using forward chaining to expand the original data with adding new inferred information. For example, adding the inferred relations between DB objects to RDF data. In this way, the generated RDF data contains more relations than relational DB. One advantage of this approach is that there is no need for new web agents that can use the new added rules. In other words, a normal semantic agent can make the use of these rules and consume the new added data in the same way as the original data without change its behavior. However, adding new data to the original one increases the size of data. However, size of data should be in a reasonable range that does not affect the web agent performance (Minsu, 2004).

The other approach to use the defined rules is using these rules during the processing of original data to infer more data on the fly without storing the new data. This approach keeps the size of the original data. However, there is overhead processing of using inference rules during searching or processing the original data. The proposed system supports both approaches.

3.3 Dumping RDF Data

The process of dumping or generating RDF data corresponding to DB contains two steps. The first step is automatic generation for RDF triples that represent the relational DB. This step is based on the mapping between DB schema and ontology. The second step is applying the user-defined rules on the generated RDF and adding the inferred data to the original RDF.

3.3.1 Dumping Relational DB into RDF

This process auto-generates RDF data corresponding to the data stored in the DB. Moreover, the proposed system dumps RDF data based on the mapping file generated by the developed mapping tool. The following steps should be executed to generate RDF data.

1- From the mapping file, get all tables mapped to ontology classes.
2- For each table
   a) Create an SQL select query to retrieve all data in the table
   b) For each retrieved record, create an instance of the corresponding ontology class of the current table. // *uri of the created instance is constructed from the following pattern (table name/ auto-increment number). i.e papers/23.*
   c) For each mapped field belongs to this table in mapping file, create an instance of the corresponding property inside the created class instance
   d) Assign a value to the created property from the retrieved data.
   e) If the field represents a foreign key, the value of the created property will be a reference to another class instance

This algorithm is implemented using Java. It generates the corresponding RDF of a DB including the relations between DB objects depending on mapping schema file.

3.3.2 Adding Extra Inferred RDF Data

Using the user-defined rules, our approach inferred additional RDF triples. These triples are added to RDF data that represents DB. Rule syntax that facilitates the process of inferring and adding extra RDF is adopted. The decided format quoted from SPARQL syntax. As a result, it is easy to use SPARQL engine in inference process.

Furthermore, the proposed algorithm for inferring extra RDF data uses forward chaining to fire the rules. This means that if the condition part of a rule is true based on the available RDF data then the action part should be inserted as a new RDF triple into the RDF data. The algorithm of adding RDF triples based on the user-defined rules is as follows.

Inputs: RDF data, user-defined rules
Output: new RDF data

For each user-defined rule

1- Get condition part of the rule
2- Construct a SPARQL select query
3- Execute the SPARQL query on RDF data
4- Replace variables in the action part of the rule with the values from the query result
5- Construct a SPARQL update query using the action part
6- Execute the update query to insert the new information to the RDF data.

This algorithm takes RDF data that represents DB and the extra rules as inputs and adds inferred RDF triples to RDF data based on rule execution. The second step in the above algorithm constructs a SPARQL query from the condition part of the current rule. The query construction process is simple in which, the common variables in the condition part and action part of the rule are extracted and a select query for these variables is constructed with the same conditions stated in condition part of the rule. The

variables in the action part are replaced with the resulted values. In addition, a new SPARQL query (insert query) is constructed from action part after replacing the variable with its values. The new query adds inferred data to the RDF data.

Moreover, the process of adding discovered relations in the proposed approach is simple and powerful. This process implemented as execution of two SPARQL queries: a select query to check rule conditions, and an insert query to execute the action part of the rule. These two queries are constructed directly based on the adopted SPARQL syntax rule format. Consequently, adding discovered relations to RDF data is easy to implement and can be executed in a high performance way based on SPARQL engine.

4 SEARCHING RDF

The proposed engine is an extension to the normal SPARQL engine. An inference step is added to the SPARQL engine to make the use of the generated RDF data and the user-defined rules. The proposed engine can answer some queries that cannot be answered using normal engines. In other words, sometime the dataset contains the desired result but the engine cannot extract it. Using inference, the engine goes behind the raw data to find query answer.

The proposed engine is developed based on Jena SPARQL API with additional inference step. The actual usage of the added inference step is not to infer more data. However, the inference step is used for query expansion process to get more detailed queries that can be answered using the normal engines.

The user can custom the engine behavior though options panel. The user determines when the engine should apply inference and to what extent. For example, the engine can be adapted to run inference only in case of no result returned by normal search. In addition, the user may stop the inference whenever a result comes. The default is that the engine will get all possible solutions.

4.1 Query Expansion

The proposed approach applies backward chaining in query answering process for query expansion. The algorithm of SPARQL query expansion is a recursive algorithm that gets all possible queries based on a set of predefined rules. Indexing for the defined rules are established to link different rules based on the inferred relations. This index for the published rules facilitates finding the appropriate rules to expand a SPARQL query. The basic idea of this query expansion is to replace query condition with other conditions based on backward chaining of the rules.

Query expansion based on backward chining algorithm is as follows:

Input: SPARQL query, rules
Output: list of new queries equivalent to the inputted query

- Get list of properties (predicates) used in query conditions.
- Get related rules that can be used to expand the inputted query (based on rule index)
- For each rule in the related rules list
  - Match between rule actions and query conditions
  - Bind matched variables and keep them in a mapping state
  - Replace the matched query conditions with rule premises
  - Recursive call to expand the new query // *this call starts matching the new query and only the rest of rule set*
- Combine all new queries in one list

A mapping state holds the mapping between different matched objects in order to construct a proper query that gets the answer of the user query. The resulted queries are executed using the normal SPARQL query engine. The results of these queries are combined and sent back to the client.

5 EXPERIMENTS

In these experiments, a large DB is converted to RDF using the proposed approach. Moreover, a set of queries are tested to show the effectiveness of adding rules to the original dataset. In addition, we applied both ways of using user-defined rules. A comparison between both approaches is presented.

5.1 Converting DB to RDF

A prototype for the proposed approach of converting DB to RDF is implemented using C#. In this experiment, the proposed approach applied on a large DB, International Semantic Web Conferences (ISWC) DB, which contains information about papers and authors involved in some conferences related to semantic web field. Figure 3 shows the schema of this DB. The total numbers of records in ISWC DB is 11213 records. It contains information

about 2595 authors and more than 1100 published papers.

The first step to convert ISWC DB to RDF is to map between DB schema and ontology. In this mapping, we used eight different ontologies:

rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
foaf=http://xmlns.com/foaf/0.1/
iswc=http://annotation.semanticweb.org/iswc/iswc.daml#
rdfs=http://www.w3.org/2000/01/rdf-schema#
dc=http://purl.org/dc/elements/1.1/
swrc= http://swrc.ontoware.org/ontology#
swc=http://data.semanticweb.org/ns/swc/ontology#
owl= http://www.w3.org/2002/07/owl#

```
<DB>
 <bridge_table name="rel_person_paper">
  <foreignkey field="PersonID"
belongToClass="InProceedings" mapToProp="Creator"
refToClass="persons" corespondFK="PaperID"
ontoIndex="dc" />
 </bridge_table>
 <table name="papers" RTClass="InProceedings"
ontoIndex="iswc">
  <primarykey>
   <field name="PaperID" />
  </primarykey>
  <foreignkey name="Conference"
RTProperty="conference" RTTable="conferences"
ontoIndex="iswc" />
 <field name="Title" RTProperty="Title" ontoIndex="dc" />
  <field name="Abstract" RTProperty="Abstract"
ontoIndex="dc" />
 <field name="Year" RTProperty="Date" ontoIndex="dc" />
 </table>
….
```

Figure 5: Mapping between DB and ontology.

A part of the mapping result is shown in Figure 5. In this mapping, the DB table *papers* is mapped to *Inproceedings* class in *iswc* ontology. Fields of *papers* table are mapped to ontology properties as shown in Figure. 5. For example, the field *title* in the table papers mapped to *Title* property in Dublin Core (*dc*) ontology.

In addition, we add user-defined rules to the mapping file to be used as an extension to the original data. According to the meaning of ISWC DB and the queries to be asked, we added some rules to the mapping file. The following rules are added during the experiment.

1. If a person A is an author to a paper Y, and a person B is an author to the paper Y then → A knows B.
2. If a person A is an author to a paper Y, and the main subject of Y is T → A is interested in T.

The first rule adds *knows* relation to the DB. *Knows* is a relation in foaf standard ontology that relates two people. ISWC DB does not contain relations between people. The second rule adds author's *interest_points* relation, which relates between person and topic.

The next step after mapping DB to ontology and adding rules is auto-generation for RDF triples represent the DB. The developed tool, figure 2, is used to generate RDF triples. The total number of generated RDF triples is 29703. This conversion process takes 7.757 seconds. Moreover, by applying the algorithm of adding inferred data to RDF, more relations are added to the original data. The number of inferred RDF triples is 18063 using the previous two rules. Execution time of the inferring and adding new triples is 39.440 seconds

A large number of RDF triples represent inferred relations are added to the original data. This extra data improves query answering process and enables web agents to get implicit information.

Our approach provides conversion from relational DB to RDF in an efficient way within reasonable execution time. In addition, it uses extra user-defined rules to generate more RDF data. Finally, our approach generates more information represented into RDF that helps semantic search engines to answer more queries.

5.2 Querying RDF Data

We prototype the query engine proposed in section four using java and Jena API (http://jena.sourceforge .net/ARQ). The generated RDF data were queried using different SPARQL queries to test different cases. Table 1 shows some queries that are used in this experiment. In this experiment, we run these queries three times against different datasets. The first is the RDF data that represents relational DB. The second dataset is the same as the first with additional inferred RDF triples. The last experiment runs the queries against the RDF that represents original DB and using the user-defined rules to expand the user queries. Table 2 shows the results of these experiments.

For example, the following SPARQL query, Q1, asks about people who know Prof. *Evgeniy Gabrilovich*.

Table 1: List of queries used in the experiment.

Number	Query
Q1	Who knows Prof. *Deepa Arun Paranjpe*
Q2	Who is interested in *Semantic search*
Q3	Get all papers in *Semantic web* topic
Q4	Who is interested in *ranking for search* and knows *Evgeniy Gabrilovich*
Q5	Get all papers in *WWW 2010* conference
Q6	Who knows Prof. *Evgeniy Gabrilovich*

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT  distinct ?x
WHERE
 {
   ?per foaf:name Evgeniy Gabrilovich.
   ?x foaf:knows ?per .
 }
```

By running this query on the original data, no result will be returned. However, after applying our approach the query returns 20 results. This means that there are 20 people know Prof. *Edward Benson* in our dataset.

Table 2: Query result using normal technique and the proposed approach.

Dataset/ size / Query number	RDF data that represent DB (2.71 MB)		RDF + inferred triples (3.39 MB)		RDF + rules (2.71 MB)	
	Number of results	Execution time	Number of results	Execution time	Number of results	Execution time
Q1	0	0.0040	2	0.0040	2	0.159
Q2	0	0.0040	39	0.0070	39	0.037
Q3	119	0.0100	119	0.0110	119	0.012
Q4	0	0.0040	9	0.0060	9	0.385
Q5	105	0.0090	105	0.0100	105	0.013
Q6	0	0.0040	20	0.0040	20	0.227

Using the user-defined rules in SPARQL engine gets better result and improves the query answer process. Moreover, the execution time of querying the RDF with additional inferred data is almost the same as RDF dataset only. However, querying RDF with additional data gives better results. On the other hand, query RDF dataset with the added rules gives same results in a little higher execution time. The execution time of the last dataset depends on the number of expanded queries not the number of results. The last dataset saves storage space. However, the second dataset saves execution time. Finally, the proposed approach can answer some queries that cannot be answered by normal approaches.

6 CONCLUSIONS

This paper proposes an approach for converting DB to RDF. Moreover, to enable web agent to deeply understand the generated data, we propose adding user-defined rules. The added rules are very useful for query answering process. Using forward chaining the proposed approach adds inferred RDF triples to the original RDF. On the other hand, the propped system uses backward chaining for query expansion and run these queries on the original dataset that represents the DB. The experiments show the effects of the proposed approach in answering queries. Moreover, the effects of using both approaches (adding inferred data and using rule in the querying process) are shown in the experiments.

REFERENCES

T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web,” *Scientific American*, Vol. 284, No. 5, 2001, pp. 34-43.

Siegfried Handschuh, Raphael Volz, Steffen Staab, Annotation for the Deep Web, *IEEE Intelligent Systems*, v.18 n.5, September 2003, pp.42-48.

Zhuoming Xu, Shichao Zhang, and Yisheng Dong, Mapping between Relational Database Schema and Owl Ontology for Deep Annotation, WI'06: *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, IEEE Computer Society, 2006, pp. 548-552.

Mamdouh Farouk, Samhaa R. El-Beltagy, Mahmoud Rafea, "On-the Fly Annotation of Dynamic Web” *Proceedings of the First International Conference on Web Information Systems and Technologies* (WEBIST 2005),” Miami (USA), may 2005, pp 327-332.

Svihla, M., Jelinek, I.: The Database to RDF Mapping Model for an Easy Semantic Extending of Dynamic Web Sites. *Proceedings of IADIS International Conference WWW/Internet*, Lisbon, Portugal, 2005, pp.27-34

Pan, Z. and Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries, In Workshop on Practical and Scaleable Semantic Web Systems, *The 2nd International Semantic Web Conference* (ISWC2003) (2003).

Ismael Navas Delgado, Nathalie Moreno Vergara, Antonio C. Gomez Lora, María del Mar Roldán García, Iván Ruiz Mostazo, José Francisco Aldana Montes: “Embedding Semantic Annotations into Dynamic Web Contents”. *Proceeding of 15th international workshop on database and Expert Systems Applications*, 2004, pp. 231-235

Chris Bizer, and Richard Cyganiak: D2R server Publishing Relational Databases on the Semantic Web, www4.wiwiss.fu-berlin.de/bizer/d2r-server/, 2010

Holger Knublauch, James A. Hendler, Kingsley Idehen “SPIN - Overview and Motivation”, http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/, February 2011.

J. Minsu and J. C Sohn, "Bossam: An Extended Rule Engine for OWL Inferencing," *In Proc. RuleML* 2004, pp. 128-138, 2004.