Generating of RDF graph from a relational database using Jena API

Jamal BAKKAS [#1], Mohamed BAHAJ[#2]
[#] Department of Mathematics and computer science, University Hassan I, FSTS
FST Settat, Km 3, B.P.:57, 7Morocco
[1] jbakkas@gmail.com
[2] mohamedbahaj@gmail.com

***Abstract*— a great part of the existing data on the web is stored in relational databases (RDB). However, the transition from the traditional web to Semantic web requires new structuring of these data. In this context we propose a method which allows automatic extraction of data from RDB and their restructuring in the form of RDF graphs using the Jena API to make them available for the Semantic Web. This structuring is to create a model of the ontology, and to enrich it from the components of the RDB schema using different classes provided by the Jena API. Subsequently, the created model undergoes a modification using SPARQL queries. Thereafter, we proceed to a set of instantiations (individuals) of these model elements using records data of the RDB to populate the ontology. Finally, we test the preservation of the RDB semantic by interrogating the resulting ontology by SPARQL queries by analogy to querying the RDB by SQL queries.**

Keyword- Mapping, RDB, ontology RDF, SPARQL, Semantic Web

I. Introduction

The need to supply the semantic web, with the data used by the web classic, most of which is stored in RDB, gave birth to a set of research to provide a new representation of the data which allows them to be usable and understandable by robots and web agents. Thus we find works that propose methods for converting existing XML document into languages that allow the knowledge representation such as RDF/RDFS or OWL [1]-[4]. Others, interested in mapping of RDB to these languages, given that a large part of data of classic web is stored in these RDB. Among this works we can distinguish two types: the first studies are limited to the conversion at the schema level, Thus, for example, we find proposals to convert UML models to ontologies [5], [6], or extracting schemas from existing RDB by basing on the principle of "reverse engineering" [7]-[9]. The other works, lead studies covering both of schema level and data level such as [10]-[12].

Our approach described in this paper falls into the latter category; it is distinguished by the structuring of the RDF graph elements that we propose to accommodate the RDB data. It begins by creating ontology elements from tables and columns of the database, and then undergoes them to modifications by removing elements created from tables that represent associations (n, n). Next, populate the ontology created by individuals by making assertions of these elements using records data of different tables. These individuals are then connecting each other by making object property assertions using SPARQL queries and foreign key values. The last step is devoted to the validation of our approach.

What remains of this paper is organized as follows: in Section 2 we describe our approach in a general way. A more detailed description is provided in Section 3. Section 4 and Section 5 are devoted to the validation of our approach. Finally, we conclude in Section 6.

II. Method description

Our approach is to generate an ontology from an RDB while keeping the semantics present in the RDB schema and links between records at the data level. We use the catalogs to extract the RDBMS schema RDB, then using the classes provided by the Jena API, we create classes, object properties and data type properties that compose model ontology corresponding to this scheme, the creation of this model must allowed to retain the sense that present the primary keys and foreign keys. Once the model is created we proceed to a modification of the model generated by deleting classes that represent tables whose fields are at the same time primary keys and foreign keys as well as the properties of data type field as having these classes. Then, to preserve the role played by the tables represented by these classes we add, for each class deleted, the object properties directly linking between the classes to which it is linked. To convert the RDB records, we proceed to create individuals which are instances of these classes using the data of RDB records, and we link between these individuals by applying assertions of object properties by taking into account the modifications made to the model. Finally, we test the validity of our approach by the RDF Validator, which checks the syntax of generated documents, and we give some ontology querying examples through SPARL language by analogy to querying the RDB by SQL language.

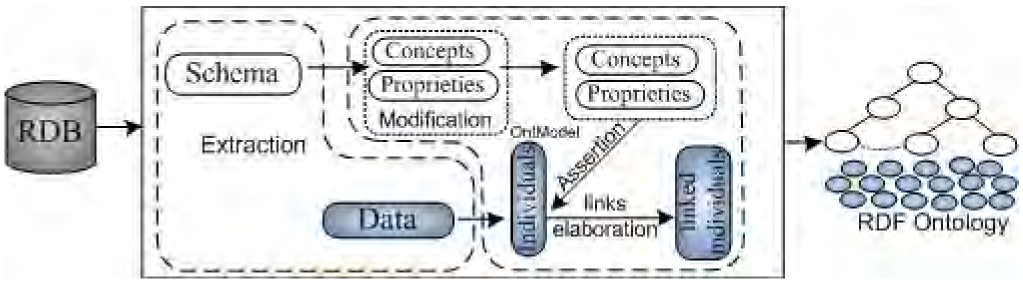


Fig1: description of the conversion method

III. MAPPING PROCEDURE

A. *Ontology preparation*

The purpose of this step is the creation and preparation of the model to accommodate the ontology classes and properties. An RDB correspond to a model, in the Jena sense, which representing the RDF graph. This model is parameterized by a namespace that is the resulting ontology URI and a prefix corresponding to this URI having the same name as the RDB.

B. *Conversion of tables*

This step focuses on the RDB schema; it allows the extraction of a table and its conversion to a class. The URI of this class is obtained by concatenating the namespace of our ontology with the table name separated by the "#" character. At this point all tables are converted to classes without distinction between tables that represent entities from those representing associations.

C. *Fields conversion*

Before starting the field’s conversion, we prepare a data type property structure which allows us to classify them, so we propose to create a super property called "ATTRIBUTE", then two sub-properties "PRIMARYKEY" and "FOREIGNKEY" who inherit the first. Thereafter, each field of a RDB table generates a data type property. Then, we associate to its domain and to its range, respectively, the URI of class corresponding to the table field, and the XSD type corresponding to the type of the field in the RDB. For the type we have created a function that made the correspondence between DBMS types and those of XML Schema. Recall that the RDF/XML language uses XML schema types for data typing.

If the field is a primary key, the corresponding property is considered as a sub-property of the "PRIMARYKEY" property , else if it is a foreign key; the generated property extends the property "FOREIGNKEY" with the generation of a object property [next paragraph], otherwise the property will extend directly from the super-property "ATTRIBUTE". All the properties that inherit the property "PRIMARYKEY" must be InverseFunctional to prevent the creation of individuals with the same value for the data type property that represents the key.

The advantage of this properties structuring, is that it allows inference engines to deduce that all the properties represent attributes since they all inherit, directly or indirectly, from the same property "ATTRIBUTE". It also allows separating properties that represent the primary keys of those representing foreign keys for eventual manipulations.

Most of the previous studies dealing this subject appoint the generated property the name of corresponding field. However, in the RDB, there can be two different fields of two tables with the same names, which will generate two properties with the same names, and we know that in ontology; data type property must have unique identifier (URI). To remedy this problem, we propose to form the URI of this property using the namespace of the current ontology and the table name concatenated with the field name. For example, the *affiliation* field, of the *Author* table, of the *Authordb* RDB is converted to a data type property with the following URI: *http://emplacement/ Authordb.RDF# author-affiliation.*

Example:

Author (<u>idAuthor</u>, nameAuthor, affiliation, #idCountry)

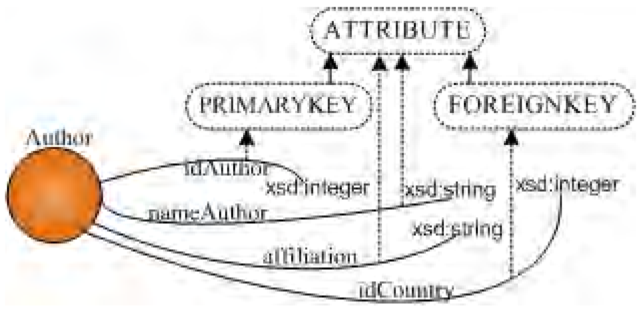


Fig2: the hierarchy of data type properties

D. *The semantics of foreign keys*

The field which is a foreign key in its table generates an object property with the URI of class corresponding to the field table as domain name, and URI of class corresponding to referenced table as range name. To ensure the uniqueness of the property, we propose to train its name by concatenating the class names representing its domain and its range like this: "domaineName_rangeName". The object properties that represent foreign keys which are not primary keys in their tables, i.e. the key represents a Association (1, n) These properties are declared as FunctionalObjectProperty to prevent the multiple assertions of this property for the same individual.

E. *Model correction*

The objective of this step is to make corrections to previous steps by deleting ontology classes that represent tables whose fields are all foreign keys and compose all the primary key of the table, i.e. that the table is an association (n, n) which is not a data carrier. In this step we interrogate our ontology with SPARQL queries and create a small segment of ontology from query results. This segment contains all classes whose all data types property inherit from the property "PRIMARYKEY" and from the property "FOREINGKEY" at the same time, as well as all object properties and data type properties whose the domain is one of these classes. These classes are replaced by object properties directly linking the classes to which they are connected without any cardinality constraint (associations (n, n)). Once created, the segment is removed from our ontology. If a class is a table that contains another field that is not a foreign key, i.e. the table represents an association (n,n) but it is data carrier, it is not affected by this correction.

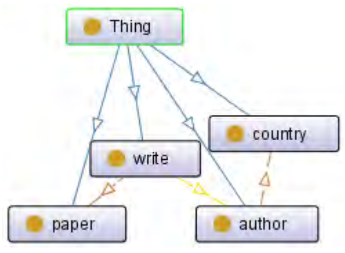


Fig 3 (a) the ontology classes before the correction

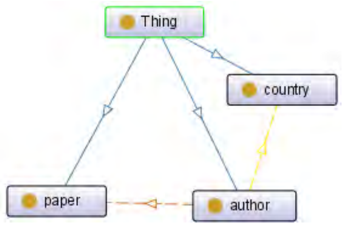


Fig 3 (b): the ontology classes after the correction

F. *Record conversions*

The purpose of this step is to populate the ontology by individuals; it consists in extracting records from a given table. And for each record we create an individual (or assertion) whose type is the class that represents the record table. To ensure the uniqueness of resources, we form the URI of the individual by the concatenation of the name space and the name of the table to which belongs the record by adding the number of the record in the table at the end. Once the individual is created, we browse the field values of the corresponding record; each value is used for the assertion of the data type property that corresponds to the field.

G. *Connecting individuals*

To reflect the role played by foreign keys in the RDB data, which is the linking between records, we apply SPARQL queries on the ontology resulting to locate individuals to connect to each other. Indeed, each individual represents a record of a table which has a foreign key that is not a primary key, we perform a SPARQL query using the value of this key to locate the individual that represents referenced record, and this individual is used for the assertion of the object property for binding both individuals together.

For example, the following SPARQL query used to locate an individual of type author and having an identifier given

```
PREFIX authorsdb: <C:/Users/Jamal/Desktop/authorsdb.RDF#>
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:       <http://www.w3.org/2002/07/owl#>
   SELECT  ?individu
   FROM           <C:/Users/Jamal/Desktop/authorsdb.RDF>
   WHERE{
         ?individu          rdf:type           authorsdb:author .
         ?individu          authorsdb:author-idAuthor           ?id
                  filter (?id=1111)
   }
```

Fig 4: Example of SPARQL query

For a table with all the fields are both primary and foreign keys, i.e. the table represents an association (n, n) which does not carry data, we parse records from the table, and for each record we use its data to formulate SPARQL queries in order to locate individuals that represent referenced records in order to link them to each other. Recall that the classes that represent these tables are deleted from our ontology [paragraph 5].

IV. RESULTS VALIDATION

To implement our approach we developed a Java application using the classes and interfaces provided by the Jena API [13] to convert relational database hosted in the DBMS MySQL.

A. Case Study

Consider the database having the following schema:

Country (idCountry, nameCountry)

Author (idAuthor, nameAuthor, affiliation, #idCountry)

Paper (idPaper, titlePaper, year)

Write (#idAuthor, #idPaper)

The database is populated with data as follows:

IdAuthor	nameAuthor	affiliation	IdCountry
1110	BAHAJ Mohamed	U. H.P / FSTS	3
1111	BAKKAS Jamal	Université Hassan I, FSTS	3
1112	SOUKLABI Abdelatif	Université Hassan I, FSTS	3
1113	JOHN Johnson	University of Paris	2

idCountry	nameCountry
1	U.S.A.
2	France
3	Morocco

idPaper	idAuthor
2220	1110
2220	1111
2221	1110
2221	1111
2223	1110
2223	1113
2224	1112

IdPaper	titlePaper	year
2220	Direct migration method of RDB to ontology while …	2013
2221	Automatic conversion method of class diagrams to...	2012
2223	Mapping RDB to RDF	2010
2224	Load Balancing Management by Efficient Controlling	2012

Fig 5: RDB to convert

Passing the RDB above [Fig 5] as input to our prototype, we obtained a graph RDF as output, this graph is visualized by Protégé using the OntGraph pluging, which gave the graph below [Fig 6], this figure shows the resulting RDF graph of ontology with two levels; the model level and assertions level, and we see clearly the links between elements of the same level.

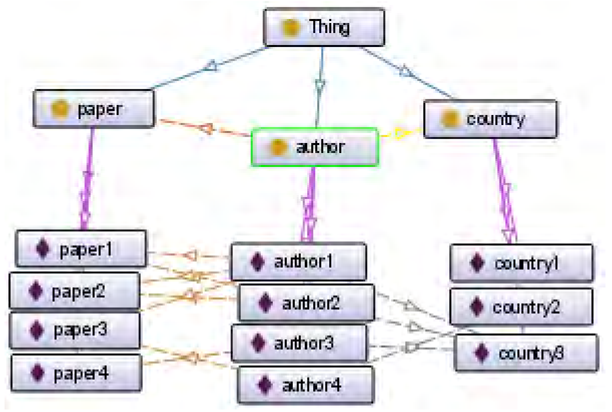
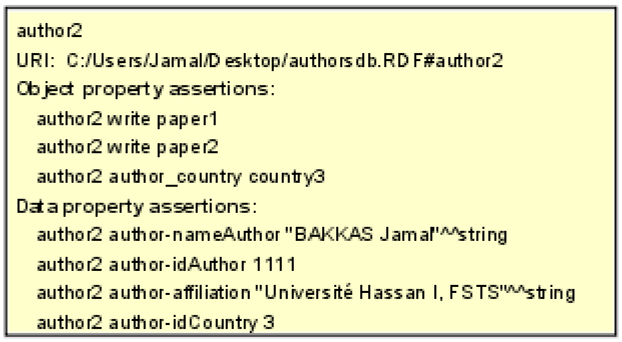


Fig 6: the OntoGraph schema of resulting RDF graph

Viewing an individual under Protégé we note that an individual is composed of: the URI which identifies it of others, assertions of data type property, and possibly the assertions of object properties. For example, the following individual is the second record of the *author* table [Fig.7]



```
author2
URI: C:/Users/Jamal/Desktop/authorsdb.RDF#author2
Object property assertions:
   author2 write paper1
   author2 write paper2
   author2 author_country country3
Data property assertions:
   author2 author-nameAuthor "BAKKAS Jamal"^^string
   author2 author-idAuthor 1111
   author2 author-affiliation "Université Hassan I, FSTS"^^string
   author2 author-idCountry 3
```

Fig 7: Example of individual generated from a record

B. *Syntactic validation*

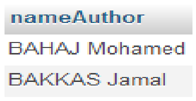
RDF Validator is a tool that allows testing the syntax of RDF/XML documents that are passed as parameters and displays a tabular presentation and graphical of this documents. To test the validity of the results generated by our system, we use the RDF validator available on the following website: http://www.w3.org/RDF/Validator/. All RDF / XML documents generated by our system from different RDB are validated by this validator.

C. *Verifying the preservation of semantics with SPARQL*

The last step is to give some examples of SPARQL queries to interrogate the resulting ontology by analogy to the interrogation of the RDB by the SQL queries. For example, to obtain the names of the authors who wrote the paper having the identifier "2221". In RDB, we must pass through the "*write"* table which represents the association (n, n). The corresponding SQL query is:

```
SELECT nameAuthor
FROM authorsdb.author, authorsdb.write, authorsdb.paper
WHERE
authorsdb.author.idAuthor = authorsdb.write.idAuthor
AND authorsdb.write.idPaper = authorsdb.paper.idPaper
AND authorsdb.paper.idPaper =2221
```

The result returned by the MySQL DBMS is as follows:



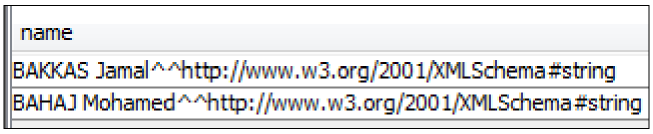
nameAuthor
BAHAJ Mohamed
BAKKAS Jamal

Fig 8: the result returned by DBMS

In Ontologies, it is sufficient to apply an assertion of the *write* object property, which ensures "Junction" between classes. Thus, the SPARQL corresponding query is as follows:

```
PREFIX authorsdb: <C:/Users/Jamal/Desktop/authorsdb.RDF#>
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>
          SELECT           ?name
          FROM    <C:/Users/Jamal/Desktop/authorsdb.RDF>
          WHERE{
                    ?auteur     rdf:type          authorsdb:author.
                    ? auteur    authorsdb:author-nameAuthor    ?name .
                    ? auteur    authorsdb:write                ?papier  .
                    ? papier    rdf:type                   authorsdb:paper.
                    ? papier    authorsdb:paper-idPaper    ?idP .
                    filter(?idP=2221)
          }
```

We used Twinkle[16] which is a SPARQL query engine for querying generated ontologies, thus, for the previous query, the result returned by Twinkle is as follows:



name
BAKKAS Jamal^^http://www.w3.org/2001/XMLSchema#string
BAHAJ Mohamed^^http://www.w3.org/2001/XMLSchema#string

Fig 9: the result returned by Twinkle

V. CONCLUSION AND FUTUR WORK

The results that we obtained by passing several RDB to our prototype show the accuracy and performance of our approach. Indeed we have shown throughout this paper that this approach preserves the data of different records of the RDB, and represents explicitly the semantics implicit presented by foreign keys. We have also shown through an example, the possibility to find for a SQL query its corresponding SPARQL query. One can therefore say that we are proposing a feasible and effective approach. Our future work will address the storage of ontologies in the RDB; indeed, ontologies, increasingly voluminous, begin to appear, calling into question their storage ways based, so far, on documents XML in his majority. The objective of this work is to improve the existing proposals or suggest others in order to ensure an ontologies storing in RDB while keeping their specificities, exploiting storage performance offered by the DBMS.

VI. REFERENCES

[1] I. Bedini, C. Matheus, P. F. Patel-Schneider1, A. Boran, B. Nguyen, "*Transforming XML Schema to OWL Using Patterns*", icsc, pp.102-109, 2011 IEEE Fifth International Conference on Semantic Computing, 2011

[2] I. Bedini, G. Gardarin, B. Nguyen, "*Deriving Ontologies from XML Schema*", In Proc. of EDA 2008 Vol. B-4, 3-17, 2008

[3] T.Rodrigues, P.Rosa, J.Cardoso: "*Mapping XML to Exiting OWL ontologies*", International Conference WWW/Internet 2006

[4] C.Cruz, C.Nicolle: "*Ontology Enrichment and Automatic Population from XML Data*", In Proc. of ODBIS 2008.

[5] D. Gasevic, D. Djuric, V. Devedzic, V. Damjanovi, "*Converting UML to OWL ontologies*". In Proceedings of the 13 th International World Wide Web Conference, NY, USA, pp. 488-489. 2004

[6] J. BAKKAS, M. BAHAJ, "*Automatic Conversion Method of Class Diagrams to Ontologies Maintaining Their Semantic Features*". In IJSCE, V-2, Issue-6, January 2013

[7] A. Imai and S. Yukita, "*RDF model and relational metadata,*" In Proc. of the International Conference on Advanced Information Networking and Applications, Xi'an, , pp. 534-537, Mar. 2003.

[8] Fan, Xuan, Pingjian Zhang, and Juanjuan Zhao. "*Transformation of relational database schema to Semantics Web model.*", Communication Systems, Networks and Applications (ICCSNA), 2010 Second International Conference on. Vol. 1. IEEE, 2010.

[9] Choi, Mi-Young, & al. "*The RDFS mapping for recursive relationship of relational data model.*" Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on. IEEE, 2010.

[10] Chen, Lei, and N. Yao. "*Publishing linked data from relational databases using traditional views.*" Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. Vol. 6. IEEE, 2010

[11] S. Zhou, H. Ling, M. Han, "*Ontology Generator from Relational Database Based on Jena*". In Computer and Information Science. Vol. 3, No. 2; May 2010.

[12] C. Bizer and R. Cyganiak, "D2r server-publishing relational databases on the semantic web." 5th international Semantic Web conference. 2006

[13] The Jena website. [Online]. Available: http://jena.apache.org/index.html

[14] S. Harris, Garlik, A. Seaborne, "*SPARQL 1.1 Query LanguageW3C Proposed Recommendation 08 November 2012*", The W3C website. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query

[15] Dodds, Leigh, "*Twinkle: A SparQL Query Tool*", the Twinkle website.[Online]. Available: http://www.ldodds.com/projects/twinkle/