Advanced Set of Rules to Generate Ontology from Relational Database

Abdeljalil Boumlik*, Mohamed Bahaj
Department of Mathematics and Computer Science, University Hassan 1, Faculty of Sciences and Technology, Settat, Morocco.

* Corresponding author: Tel.: +212660250506; email: mohamebahaj@gmail.com
Manuscript submitted September 9, 2015; accepted November 20, 2015.
doi: 10.17706/jsw.11.1.27-43

Abstract: Nowadays, the majority of data sources in the current web are stored in Relational Data Bases (RDB), the semantic web main idea is to solve the problem of sharing and reusing information between applications and companies in different common areas, based on data stored in RDBs. This article present a complete automatic approach that generate Ontology from a giving relational database based on a set of rules that extract semantics from RDB and transform it to OWL file. Our approach treat most complicated relationship types and constraints like simple and multiple inheritance, transitive chain, disjoint, completeness constraint and N-ary relations. From other side, our solution deals also with mapping data at the same time, think that make this solution more powerful, complete and effective. Our approach composed of four processing stages, analysis, extraction, mapping and finally a verification step before generating the OWL file.

Key words: Data management, mapping RDB, ontology, OWL, RDB.

1. Introduction

With the increasing use of semantic web, many researches are interested more and more about solving important problems in this area like interoperability, data integration and information's exchange between different systems, due to the relational data base's (RDB) limitation that ignore semantics level in stored data. Therefore, most of them try to find solutions and methods that transform automatically RDBs to Ontologies for semantic web use.

The semantic web was offering the possibility to resolve such complicate problems related with interoperability, data mapping and schema structure to provide a better machine assistance for human users, by making the information process able and understandable by machines, using the concept of dynamic data, called ontologies.

There have been several definitions of what an ontology is, and we chose the one proposed by Tom Gruber that defined ontology as "a formal and explicit specification of a shared conceptualization that refers to an abstract model of some phenomenon in the world that identifies the relevant concepts of that phenomenon". In the context of database systems, ontology could be defined as a process of data abstraction and schema models that are similar to relational, conceptual and hierarchical models, which is supposed to model individuals' knowledge, attributes and relationships. Ontologies are particularly specified in languages that make possible the abstraction of data structures and allow strategies implementation. Semantic Web is then expected to provide languages that can both express data and rules

for reasoning about the data, and also to export rules from any existing knowledge-representation system into the web.

Table 1. Existing Works

Approaches	Concept	Remarks
Learning ontology from Relational Database [1]	• Methodology is automatic • Limited mapping rules discussion • No implementation	• Mapping discussed at basic level • Manual work is required
Algorithms for Mapping RDB Schema to RDF for Facilitating Access to Deep Web [2]	• Limited mapping rules discussion • Mapping Schema only • Extraction of Mata-data is performed • Ontology generate	• Mapping based on data dictionary • RDF is used
Mapping relational database into OWL Structure with data semantic preservation [3]	• Methodology is automatic • Process is divided into three parts • Extraction of Mata-data is performed • creating Canonical model	• Mapping discussed at basic level • OWL file is generated
Ontology Learning for The Semantic Web [4]	• Extracted concepts and terminologies • Not automatic • No implementation or results	• Mapping based on other tools • E-gov domain only
Relational database as a source of ontology creation [5]	• Automatic method • Limited mapping rules • No implementation	• Classic rules are discussed • Mapping data ignored
Ontology Construction from Relational Database [6]	• Not automatic • Human experts is required • Ontology generated • No implementation	• Mapping schema only • lack of many specific database cases
Efficient Semantic Information Retrieval System from Relational Database [7]	• Tool developed but not discussed • Mapping schema only • Extract knowledge from the generated Ontologies	• Mapping Schema only • Generate the Ontology from RDB • Standard rules are discussed
Schema and Data Conversion from RDB into OWL2 [8]	• Automatic approach • Tool was developed • Establishing mapping rules • Conversion of data-bases to ontology	• The used algorithms are not optimized enough • Complexes relations are not discussed (n-ary relation)
A Framework for OWL DL based Ontology construction from RDB using Mapping and Semantic Rules [9]	• Using OWL DL language • Semi-automatic • Application of mapping rules • Creation of ontology document	• Approach does no discussion on complexes mapping cases. • Tool not implemented
Ontology Based Semantic Integration of Heterogeneous Databases [10]	• Automatic • Establishing mapping rules • Conversion of data-bases to ontology	• Basic rules discussion • Constraints discussion was ignored

All The existing approaches for mapping RDBs to ontology use the schema mapping to transform the components of the conceptual data model or the physical model into ontology's concepts and relations.

In this work, we propose a complete, automatic and enhanced transformation rules that map a RDB's schema and data to Ontology Web Language file. This approach manages schema mapping and data

analysis techniques to detect inheritance, disjoint, completeness and N-ary relationship and other standard types of relationships.

The rest of this paper is organized as follows. Section 2 discuses related works and ontology approaches that cover this mapping. Section 3 describes the proposed mapping rules. Implementation and evaluation are presented in Section 4. Finally, Section 5 concludes this paper, and discusses the perspectives of this work.

2. Related Works

We can found several approaches that deal with RDB to OWL mapping, e.g. [1]-[6] but most of them contain simple and limited cases, rules, and doesn't cover most complex relations and constraints like disjoint, Completeness constraint and N-ary. Various works are limited on the schema level without taking in consideration the Data that should be mapped too, they didn't also provide a real implementation or prototype that improve their methods. In the below table we will explain the idea behind each approach with our remarks.

From the above table, we discuss most recent methods and solution that deal with this type of mapping, and we conclude that most of them have at least one of the following defects:

- Most of them are Semi-automatic solutions and needs human intervention after mapping
- They treat only RDB's Schema only, without Data.
- They are very limited on simple structures, relationships and constraints
- No implementation for their solution or algorithms explanation
- No validation of the generated ontology.

In short, the novelty of our approach consist in the capacity to map automatically the Schema and Data at the same time from an input RDB using optimized and advanced algorithms to detect and map complex relationships between tables like completeness, multiple inheritance and N-ary,…etc, think that was not treated in most existing works above.

3. Overview

In this section, we present the approach's logic and transformation flux with different functionalities we offered to the user. Fig. 1 explain the processing stages one by one before generating the OWL file:

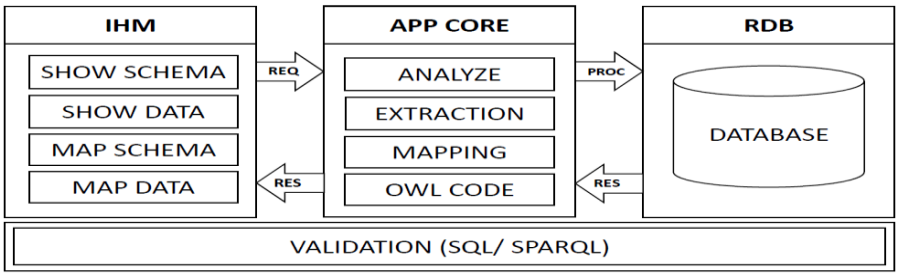

Fig. 1. Generation process of OWL code for schema and data.

Analysis stage consists to make a classification of entities type, and discover the concepts, attributes, relationships and axioms, this step provides the necessary information related to the concerned RDB

Extraction stage consists to make an extraction to the domains semantic by analyzing database schema and data instance, this process will identify and detect different type of entity tables (normal entity, weak entity, subtype entity, and super type entity, etc.), and also the binary relationship between tables like (many-to-many, one-to-many, and one-to-one etc.).

Mapping (generation) stage consists to execute the necessary transformation rules, depends on each

cases exist at our data base level before generate the OWL file, and prepare it to for validation steps.

Validation stage consists to verify the generated OWL file and execute testing queries between the generated ontology and database using SQL and SPARQL languages, then compare the obtained results.

3.1. Definition

3.1.1. Relational Database

A relational database schema (R), is a finite collection of relations (Rel). A relation consists of the name of the relation, attributes (columns) and constraints (Integrity constraints, unique constraint, not null constraint …) which restrict the data instances that can be stored in the database.
In this article we present the relational database as below:

$$R = (Rel(r), Attr(A, r), PK(p, r), FK(f, r))$$

$Rel(r)$: existing relation in R
$Attr(A)$: function returns that A is an attribute in T
$PK(T)$: function returns that A is a single or composite primary key of the table T.
$FK(T)$: function returns that A is a single or composite foreign key of the table T.

3.1.2. Ontology

Ontologies (Onto) used in this paper are expressed by OWL DL. For notation, we use (C) to represent a class, and (P) to represent a property. Further, DP denotes a datatype property and OP denotes an object property. dom(P) gets the domain(s) of P, and Rang(P) gets its range(s). We define our ontology as follows:

$$Onto = (C, P, DP, OP, dom(P), Rang(P))$$

3.2. Types of Entity Tables

We should classify different types of entity tables (normal entity, weak entity, subtype entity, and super type entity, etc.) and various relationship tables including binary relationships (many-to-many, one-to-many, and one-to-one) tables and n-ary relationship tables.

A particular table type can be detected by analysing its primary key, foreign key(s), and sometimes the instance data, as we have below:

- Normal entity: it's a relation that has only one Primary Key and no foreign key.
- Strong entity: Tables that contains only simple attributes without foreign keys, AND Tables that their primary key is also a foreign key referencing unique table
- Weak entity: it's a relation that has exactly one primary key and one foreign key, and the foreign key is a subset of the primary key.
- Many-to-many: it has exactly two foreign keys and one primary key, and the primary key is the composite of the two foreign keys.
- N-ary relationship: Means that we link an individual to more than a single individual or value to it, and has at last three foreign keys and one primary key, the primary key is the composite of the three foreign keys.
- Subtype entity: A subgrouping entities in an entity type that has attributes distinct from those in other subgroupings (new type that is similar but not identical to an already defined type).
- Super type entity: A generic entity type that has a relationship with one or more subtypes

4. Rules and Algorithms

4.1. Create Classes

Rule 1: Every normal entities that has at least one PK and no FK should be mapped to a normal Class:
Example: Professor (Prof_Id, Prof_Name, Prof_email, Address)

```
< owl Class    rdf ID = " Professor "/ >
```

Rule 2: Every normal entities that has the same Primary Key PK(A,T) and respect this condition:
IF PK (A, T1) = PK (A, T2), can be mapped to the same Class on our ontology.
Example: Staff (Stf_Id, Name, email, affectation)
StaffEx(Stf_Id, Name, email, affectation)

```
< owl Class    rdf ID = "Staff"/ >
```

4.2. Create Properties

4.2.1. Object Properties

Rule 3: Every foreign key (FK) that refer to a Primary Key in other table will be mapped into two Object-Properties (mutually inverse), with:

Domain:Current table
Range :Referenced table by the Foreign Key FK
>>>>AND REVERSE**<<<<**
Domain:Referenced table by the Foreign Key FK
Range :Current table

Example: Order (OrderId, OrderDate, OrderDetails)
OrderItem (OrderedItem,Description, #OrderId)

```
<owl:Class rdf:ID="OrderItem"/>
    <owl:ObjectProperty rdf:ID="OrderItemHAsOrder">
      <rdfs:domain rdf:resource="#OrderItem" />
      <rdfs:range rdf:resource="#Order" />
    </owl:ObjectProperty>
    <owl:InverseFunctionalProperty rdf:ID="Ordered Items">
      <rdfs:domain rdf:resource="#Order" />
      <rdfs:range rdf:resource="#OrderItem"/>
    <owl:inverseOf rdf:resource="#OrderItemHasOrder" />
</owl:InverseFunctionalProperty>
```

4.2.2. Data properties

Rule 4: Any attributes in R that are not PK (T) nor FK (T) and that cannot be transformed to an OP (object property), should be transformed to a Data type property in our ontology.
Example: Professor (Prof_Id, Prof_Name, Prof_email, Address)

```
<owl:DatatypeProperty rdf:about="#Prof_Name ">
      <rdfs:domain rdf:resource="#Professor"/>
      <rdfs:range rdf:resource="&xsd;string"/>
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#Prof_email">
        <rdfs:domain rdf:resource="#Professor"/>
        <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="# address">
        <rdfs:domain rdf:resource="#Professor"/>
        <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

4.2.3. Mapping constraints

In relational database (R), we have several types of constraints such as Not Null, Unique, both (Not Null & Unique), for that we suggest the below treatment for each type:

4.2.4. Primary key

Rule 5: Every Primary Key exist on table (T), should be mapped as "InverseFunctionalProperty" with a "minCardinality" that should be set to **1** on the OWL property.

```
<owl:InverseFunctionalProperty rdf:ID= "PK_attribute"/>
    <owl:classrdf:ID = "Table_name">
      <rdfs:subClassOf>
     <owl:restriction>
       <owl:OnPropertyrdf:resource = "#PK_attribute"/>
              <owl:minCardinality rdf:datatype ="&xsd: nonNegativeInteger"/>
               1
              </owl:minCardinality>
       </owl:restriction>
      </rdfs:subClassOf>
 </owl:class>
```

4.2.5. Foreign key

Rule 6: Foreign Key that refer to a strong class should be mapped as an Object Property.

Example: ClassGrade (ClassGrID, name, level)

Student (Id, name, age, #ClassGrID)

The ClassGrID column on the Student table is a foreign key that refer to the ClassGrade, thus, it should be mapped as Object Property as mentioned below:

```
<owl:ObjectProperty rdf:about="#ClassGrID ">
      <rdfs:domain rdf:resource="#Student"/>
      <rdfs:range rdf:resource="#ClassGrade"/>
</owl:ObjectProperty>
```

4.2.6. Not null

Rule 7: For Not Null constraint in relation database (R), means that each tuple in the table should have mandatory value, hence it should be presented using minCardinality restriction seted to 1.

Example: Class (ClassID (**INT**), name (**VARCHAR, NOT NULL**))

```
<owl:Class rdf:about="#Class">
```

```
<rdfs:subClassOf>
 <owl:Restriction>
   <owl:onProperty rdf:resource="#name"/>
     <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
       1
     </owl:minCardinality>
 </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

4.2.7. Unique

Rule 8: For Unique constraint in relational database we suggest to set maxCardinality restriction to 1 in order to avoid any individuals having the same value.

Example: Class (ClassID, name (**UNIQUE**))

```
<owl:Restriction>
  <owl :onProperty rdf:resource=”#AttributeName/>
  <owl:maxCardinality>1</owl:maxCardinality> </owl : Restriction >
```

4.2.8. Unique and not null

Rule 9: If we have any attribute with Unique and Not Null constraints, we propose to make a combination of the above two cases and set the maximal and minimal cardinality to 1.

Example: Class (ClassID, Name **(UNIQUE, NOT NULL)**)

```
<owl : Restriction >
    <owl :onProperty rdf:resource=”#Name”/>
        <owl:minCardinality> 1</owl:minCardinality>
        <owl:maxCardinality> 1</owl:maxCardinality>
</owl : Restriction >
```

4.2.9. Mapping data

Most existing approaches map schema only, and ignore data, which is very important to improve the correctness of the generated ontology.

In our approach we propose to convert database records to an equivalent individual with the same class type, and that will contain the values of each row in the current record, below an example that illustrate our logic:

```
<owl:RecordName rdf:ID=“TableName_PK[T]”>
  <owl:type      rdf:resource="#CurrentTableName"
    <colomun1     rdf:datatype=”&xsd:column’sType”>Value</column1>
    <colomun2     rdf:datatype=”&xsd:column’sType”>Value</column2>
    <colomun[i] rdf:datatype=”&xsd:column’sType[i]”>Value</ colomun[i]>
<owl:RecordName>
```

For Foreign attributes exist on the current table they should be presented as below format

```
    <CurrentTable_ReferncedTable   rdf:resource=’ReferencedTableName’ >
```

4.2.10. Mapping disjoint relation

Disjoint relation can be identified using constraint (PK/FK) between three tables, two of them are Sub-type tables that are related to a common Super-Class type.

Rule 10: If we have three tables T1, T2 and T3 with no-direct relation, and we have a foreign key relation between table T1 and T2, and another foreign key relation between table T2 and T3, with no relation (PK or FK) between T1 and T3 then we said that we have a Disjoint relation, and should be mapped on our OWL file as below:

Example: Student (StudentID, Name, email, address)

GraduateStudents (Degree, position, #StudentID)

UndergraduateStudents(CurrentClass, Speciality, #StudentID)

```
<owl:Class rdf:about="#GraduateStudents">
    <rdfs:subClassOf rdf:resource="#Student"/>
    <owl:disjointWith rdf:resource="#UndergraduateStudents"/>
</owl:Class>

<owl:Class rdf:about="#UndergraduateStudents">
    <rdfs:subClassOf rdf:resource="#Student"/>
    <owl:disjointWith rdf:resource="# GraduateStudents"/>
</owl:Class>
```

4.2.11. Mapping transitive chain relation

Rule 11: This relation can be detected when we have three tables T1, T2 and T3, and if we have a foreign key relation between table T1 and T2, and another foreign key relation between table T2 and T3, then we said that we have a Transitive chain relation between table tables T1 and T3 and mapped as below:

Example:

```
<owl:ObjectProperty rdf:ID="PK(A,T)">
    <rdf:type rdf:resource="owl;TransitiveProperty"/>
       <rdfs:domain rdf:resource="#T1" />
       <rdfs:range rdf:resource="#Tx" />
</owl:ObjectProperty>
```

4.2.12. Mapping completness constraint

Rule 12: This relation specify that every entity in the Superclass must be a member of at least one Subclass in the specialization. For example, if every PERSON must be either an Student or a EMPLOYE, then the specialization {STUDENT and EMPLOYE} of the below example is a total specialization of PERSON

Example: Person (ID, Name, address, e-mail)

Student (#ID,StudentName, Class)

Employe ((#ID,Employe_Grade, speciality)

Engineer ((#ID,departement, speciality)

```
<owl:class rdf:ID="Person">
   <owl:unionOf rdf :parseType="Englobe">
       <owl:class rdf:about="#Student" />
       <owl:class rdf:about="#Engineer" />
       <owl:class rdf:about="#Employe" />
   </owl:unionOf>
</owl:class>
```

4.2.13. Mapping n-ary relation

N-ary relation is a complex type of relation in which we link an individual to more than a single individual or value. Researchers confirm that the N-ary relation is a difficult case that can be represented on Ontologies, due to his structure that links only binary relations between classes. This binary relations can be represented through object properties. Contrariwise, N-ary relation cannot be presented in the same way.

In order to handle this case, W3C propose two different solutions to deal with this N-ary relation, the first one consist to create an individual that present the relation itself with links to the instance then to the participating tables, thus, human intervention will be necessary to choose the subject of the relation (cannot be managed automatically), For that, the second solution proposed in Fig. 2 is most adaptable and match our automatic approach. This solution consist to create an individual to represent the relation instance with links to all participants tables, that means our algorithm will detect the bridge table that links the participating tables in the N-ary relation and transform it to a Bridge class in OWL with restrictions (allValuesFrom or someValuesFrom) depend on the participation level of the concerned tables. The example presented in Fig. 3 will explain our algorithm and the mapping process too:

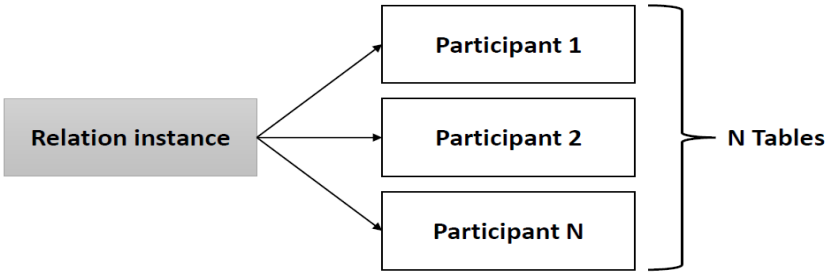

Fig. 2. Automatic mapping of n-ary proposed by W3C.

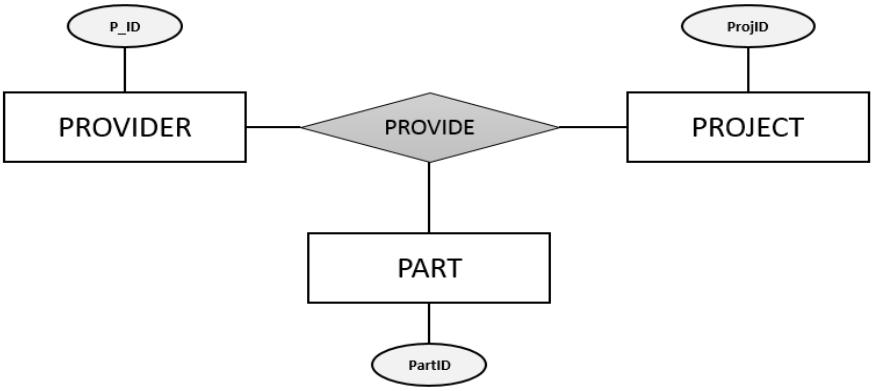

Fig. 3. Example of an n-ary relation.

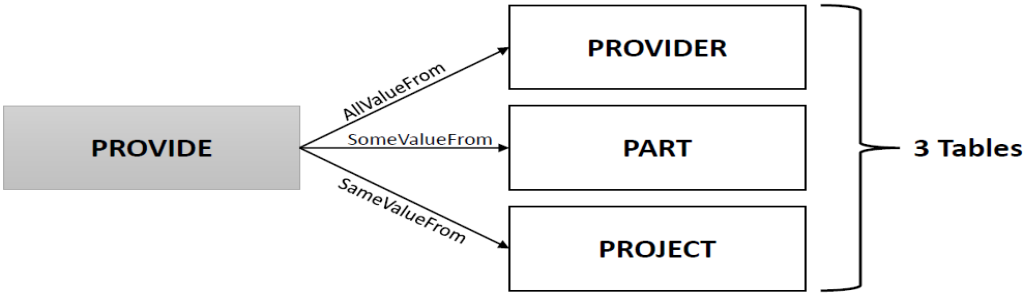

Fig. 4. N-ary mapping solution.

Example: Provider (P_ID, Company_name, address, Company_type)
Project (ProjID, Project_name,)
Part (PartID, Part_description)
Graphical presentation

The solution above consist to create Bridge Class (PROVIDE) that regroup the classes participating in this N-ary relation (PROVIDER, PART, and PROJECT) with restriction attribute 'AllValueFrom' if all records of the participating tables are referenced in the bridge table, Else a 'someValuesFrom' restriction is used, a simple illustration use mentioned in the Fig. 4 :

Mapping Result

```
<owl:Class rdf:ID="PROVIDE_RELATION">
    <rdfs:subClassOf>
        <owl:Restriction>
         <owl:someValuesFrom>
              <owl:Class rdf:about="#PROVIDER"/>
          </owl:someValuesFrom>
          <owl:onProperty>
             <owl:ObjectProperty rdf:about="has_Provider"/>
          </owl:onProperty>
        </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
      <owl:Restriction>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#PROJECT"/>
             </owl:someValuesFrom>
             <owl:onProperty>
                <owl:ObjectProperty rdf:about="has_Project"/>
             </owl:onProperty>
       </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:allValuesFrom>
                <owl:Class rdf:about="#PART"/>
            </owl:allValuesFrom>
            <owl:onProperty>
                 <owl:ObjectProperty rdf:about="has_Part"/>
            </owl:onProperty>
         </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Algorithms

In this section, we present our algorithms that deal with each rules mentioned above. The main procedure start with converting tables, constraints, standard and specific relations and also the Data we have in the RDB.

```
Procedure MappingDatabase(S)
Input: Schema S
Begin
   MappingTables(S)
   MappingConstraints(S)
   MappingTransitivechain(S)
   MappingDisjointRelations(S)
   MappingCompletenessConstraints(S)
   MappingData(T)
End
```

4.3. Mapping Algorithm for Tables

Referring to the **Rule 1**, the algorithm will convert every normal relation to an owl class, as you can see below:

```
Procedure MappingTable(T)
Input: Schema (S), Table (T)
Output : Class C
Begin
    For (∀(T) in Schema (S))
      loop
        CreateClass:<owl:Class rdf:ID="Ti" />
      End loop;
    End For;
End;
```

4.4. Mapping Algorithm for Constraints

In the below algorithm we will respect all **Rules 3, 4, 5, 6 , 7** and **8** related to different types of constraints as Primary Key, Foreign Key, Not Null and Unique that we see below :

```
Procedure MappingConstraints(S)
Input Schema (S), Table T, Referenced Table (RT), Attribute (Attr), PK(A,T), FK(A,T), Not
Null   NotNullCheck(A,T), Unique Unique(A,T)
Begin
 For ∀ (T) in Schema (S) loop
   For ∀(attr A) in Table (Ti) loop
     /*Mapping Primary Key*/
     If(PK(Aj,Ti) = true) Then
       Call Function MapPrimaryKeyOfTable(Ti) //Refer to Rule5
     /*Mapping Foreign Key*/
     Else if(FK(Aj,Ti) = true) Then
       Call Function MapForeignKeyOfTable(Ti) //Refer to Rule6
     /*Mapping Unique && Not null Constraint*/
     Else if((Unique(Aj,Ti) = true) && (NotNullCheck(Aj,Ti) = true)) Then
     Call Function MapUniqueAndNotNullAttrOfTable(Ti) //Refer to Rule 9
     /*Mapping Unique Constraint*/
     Else if(Unique(Aj,Ti) = true) Then
     Call Function MapUniqueAttrOfTable(Ti) //Refer to Rule 8
     /*Mapping Not Null Constraint*/
     Else if(NotNullCheck(Aj,Ti) = true) Then
     Call Function MapNotNullAttrOfTable(Ti) //Refer to Rule 7
     End If;
   End Loop;
 End Loop;
End;
```

4.5. Mapping Algorithm for Transitive Chain

Transitive chain relations is identified using Primary and foreign keys between three tables that have no direct relation as described in paragraph (Mapping transitive chain relation), below is the proposed algorithm:

```
Procedure Transitive (T, Tx)
   Input Table T1, Attr A, PK(A,T), FK(A,T)
   Begin
      For(∀ Attr(A) in(Tx) )Loop
        If(FK(Ai,Tx))= true) then
          FK(Ai)= FK(Ai,Tx)
            For ∀ (T) in Schema (S) Loop
```

```
                    If(PK(Fk(Ai), Ti)= true && Ti <> T1)then
                      CreateTransitiveRel (T, Ti)
                    End If;
                End Loop;
            End If;
        End Loop;
    End;
```

4.6. Mapping Algorithm for Disjoint Relation

In this part we present disjoint relation algorithm, which extract all tables responding to rule 10

```
procedure Disjoint(S)
Input FKS List of foreign keys of all Tables in S
        PKS List of primary keys of all Tables in S
        MAP(key,value) collection of keys and values
BEGIN
 FOR EACH Ti IN S LOOP
   FKS=getForiegnKeyOfTable(Ti)
     IF COUNT(fks)!=0 THEN
       PKS=getPrimaryKeyOfTable(Ti)
         IF PKS[i]!=FKS[i] THEN
             MAP.key=Ti
             MAP.value=fki
         END IF
     END IF
     IF getNumberOfPrimaryKey(Ti)=0 THEN
             MAP.key=Ti
             MAP.value=fki
      END IF
 END LOOP
 mapDisjoint(map)
END
```

```
Procedure mapDisjoint(map)
Input MAP(key,value)
BEGIN
 FOR EACH iti IN map LOOP
   key1=iti.key
   value1=iti.value
      <owl:Class rdf:about=\"#" + key1 + "\">
         <rdfs:subClassOf rdf:resource=\"#" +value1+ "\"/>
          FOR EACH itj IN map LOOP
             key2=itj.key
             IF key1=key2 THEN
               <owl:disjointWith rdf:resource=\"#"+key2+"\"/>
             END IF
          END LOOP
      </owl:Class>
 END LOOP
END
```

4.7. Mapping Algorithm for Completeness Constraints

The below algorithm explain our optimized solution to map completeness constraint using MAP collection function, in order to retrieve in the first step all primary and foreign keys, then we applied the necessary checks and controls before extracting relationships between tables:

```
Procedure Completeness()
Input
Schema(S)
```

```
List PKS    : a List of primary Keys
List FKS    : a List of Foreigh Keys
Map (PKS) : Collection of Primary Keys //(Key, Value)
Map (PKS) : Collection of Foreign Keys //(Key, Value)
 FOR ∀(T) IN Schema (S) LOOP
  IF( (element in PKS) = (element in FKS) AND RTi = GetRefTab(element(PKS)))
  THEN
     Map=PKS ,FKS//fulfill tables that have A as PK and A is FK in the RT Ex: [PK(T),FK(Ty)]
     Map=FKS,FKS //fulfill tables that have A as FK and A is FK in other T Ex: [FK(T),FK(Ty)]
  END IF;
 END FOR;
 CALL MapCompleteness(map) //Call below function to proceed with the mapping
END
mapTotalness(map1){
FOR ∀ (it) IN map1 LOOP //For each Iterator in MAP1 Do…
 IF(map1.cle=map1.valeur THEN
  <owl:class rdf:ID="    map1.cle    ">
   <owl:unionOf rdf:parseType="Collection">
    map2=map1 //Compare MAP 1 and MAP 2 to retrieve tables name
    FOR ∀(it) IN map2 LOOP//For each Iterator in MAP1 Do…
     IF map1.cle=map2.valeur AND map2.cle=map2.valeur THEN
      <owl:class rdf:about=" map2.cle "/>
    END FOR
   </owl:unionOf>
 </owl:class>
 END
```

4.8. Mapping Algorithm for Data

In this part, we present the algorithm we use to map Data stored in the relational database, the full details are discussed above in chapter “Mapping Data”.

```
Procedure MapData(S)
Input : Schema (S)
Tables = GetAllTablesInSchema(S)
 For each Table in Tables loop
  Results= getAllcolumnsFromTable(Ti)
  CurrentTable = Ti
  Count=1
  For each Record in Results Loop
   Print <owl:RecordName    rdf:about=\"#" +CurrentTable+ "_" +PK(Ti)+ "\"/>"
    while    getNumberColumnOfTable(Ti) is not the Last column DO
      Column_Name = getMetadataOfColumn(i)
      IF (isPK(Column_Name,Ti) == true ){
       Print <"Column_Name" rdf:dataType=&xsd;Type\">1</Column_Name">
      }
      Else If (isFK(Column_Name,Ti) != true){
       <Column_Name" rdf:dataType=&xsd;String>
         getValueOf(Column_Name
       </Column_Name>
     }
     End if
     fks = GetForeighKeyOfTable(Ti)
     IF (fks!=0){
      Print    <CurrentTable"_"ReferencedTable"rdf:resource=”#ReferencedTable/>
      }
    End Loop
    Print </owl:RecordName >"
  End Loop
END
```

4.9. Mapping Languages Comparison

Table 2. Comparative Study of Existing Methods

Approaches	[1]	[2]	[7]	[8]	[9]	[10]	Our Method
PK & FK	X	X	X	X	X	X	**X**
Unique	X	X	-	X	-	X	**X**
Not Null	X	X	-	X	X	X	**X**
Unique & Not Null	X	X	-	X	-	X	**X**
One To Many	X	X	X	X	X	X	**X**
Many-to Many	X	X	X	X	X	X	**X**
Simple inheritance	X	X	X	X	X	X	**X**
Multiple inheritance	-	-	-	-	-	-	**X**
Many-to-Many With attr	-	-	-	X	-	-	**X**
Disjoint Relation	-	X	-	-	-	-	**X**
transitive chain	-	X	-	X	-	-	**X**
Completeness	-	-	-	-	-	-	**X**
N-ary relation	-	-	-	-	-	-	**X**
Mapping Data	-	-	X	X	X	-	**X**

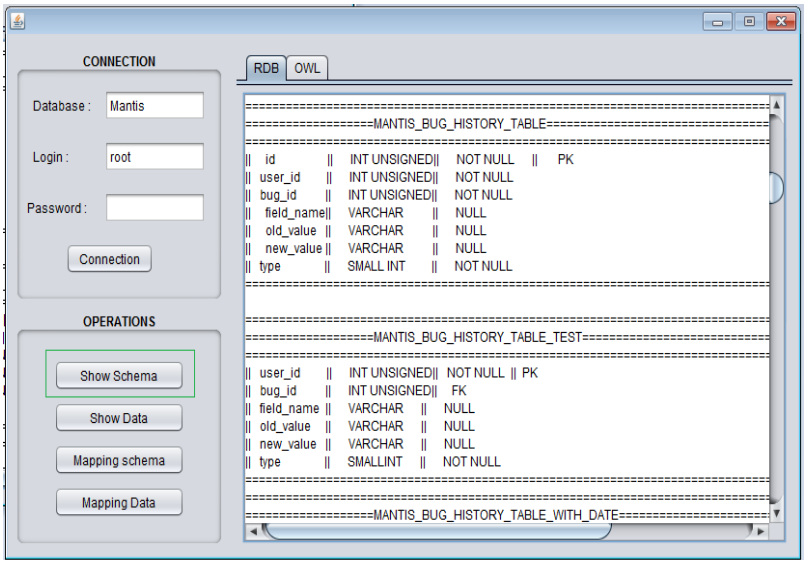

Fig. 5. Show schema algorithm.

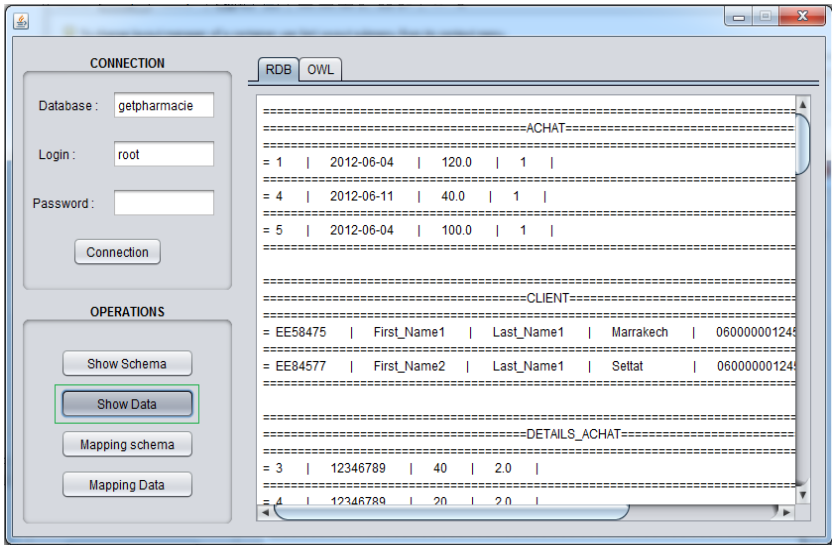

Fig. 6. Show data algorithm.

5. Implementation

To demonstrate the validity of our approach, we developed a prototype called "AdvancedOnto" that contain above algorithms. The tool was implemented using Java language and JDBC connection to the database, and can execute iterator's scripts to navigate the schema (Fig. 5) and represent data stored in concerned RDB (Fig. 6). The results we have achieved and presented in Fig. 7 and Fig. 8 shows performance and scalability of our mapping algorithms that deal with schema and data at the same time.

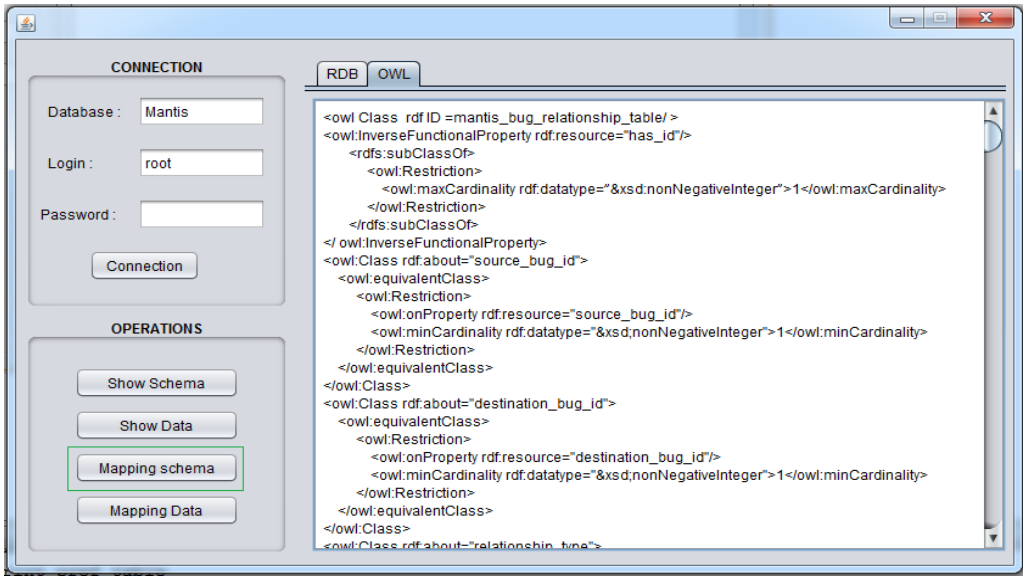

Fig. 7. Mapping schema to owl file.

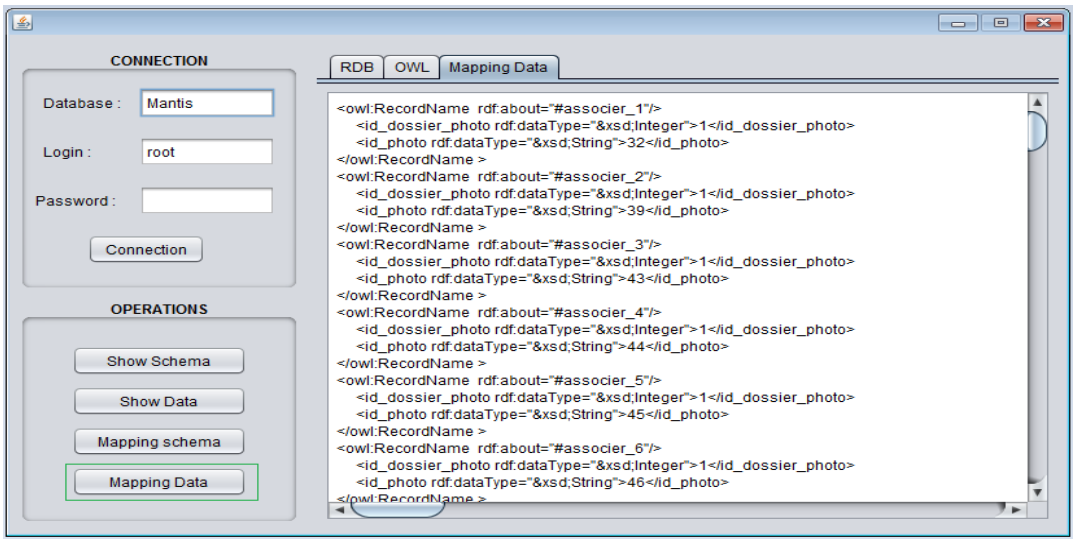

Fig. 8. Mapping data to owl file.

6. Conclusion and Future Work

In this paper, we proposed an approach based on enhanced and complex set of transformation rules to generate OWL file from relational databases, our approach is completely automatic and optimized solution comparing with all others existing solutions. We are focused on this article to deal with complicate relations between tables like completeness constraint, multiple inheritance, disjoint and N-ary relation...etc. We also handle the mapping of data, this part that was ignored in many existing approaches. Our next goal will

be focused on the triggers side to finish a framework that will allow full mapping of database to ontology, we also think about validation phases that aims to create a reverse process with a mediator in order to validate the obtained ontology with original RDB using SQL and SPARQL queries to compare the results.

References

[1] Li, M., Du, X. Y., & Wang. S. (2005). Learning ontology from relational database. *Proceedings of the 2005 International Conference on Machine Learning and Cybernetics*.

[2] Mallede, W. Y., Marir, F., & Vassilev, V. T. (2013). Algorithms for mapping RDB schema to RDF for facilitating access to deep Web. *Proceedings of the First International Conference on Building and Exploring Web Based Environments*.

[3] Noreddine, G., Khaoula, A., & Mohamed, B. (2012). Mapping relational database into OWL structure with data semantic preservation. *International Journal of Computer Science and Information Security*, *10(1)*.

[4] Alexander, M., & Steffen, S. (2005). Ontology learning for the semantic web. I*EEE Intelligent Systems*, *16(2)*, 72-79.

[5] Telnarova, Z. (2010). Relational database as a source of ontology creation. *Proceedings of the 2010 International Multi Conference on Computer Science and Information Technology*.

[6] Jaleel, A., Islam, S., Rehmat, A., Farooq, A., & Shafiq, A. (2011). Ontology construction from relational database. *International Journal of Multidisciplinary Sciences and Engineering*, *2(8)*.

[7] Rajeshkumar, T., Ramathilagam, C., & Valarmathi, M. L. (2014). Efficient semantic information retrieval system from relational database.

[8] Larbi, A., & Mohamed, B. (2014). RDB2OWL2: Schema and data conversion from RDB into OWL2.

[9] Ramathilagam, C., & Valarmathi, M. L. (2013). A framework for OWL DL based ontology construction from relational database using mapping and semantic rules applications.

[10] Kavitha, C., Sadasivam, G. S., & Sangeetha, N. S. (2011). Ontology based semantic integration of heterogeneous databases*. European Journal of Scientific Research*, *64(1)*, 115-122, 2011.

[11] Justas, T., & Olegas, V. (2007). Building ontologies from relational databases using reverse engineering methods. *Proceedings of the International Conference on Computer Systems and Technologies*.

[12] Farid, C. (2008). Mining the content of relational databases to learn ontologies with deeper taxonomies. *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (pp. 553- 557).

[13] Kobra, E., Mohsen, K., & Yanehsari, N. R. (2009). Building ontologies from relational databases. *Proceedings of the First International Conference on Networked Digital Technologies* (pp. 555-557).

[14] McGuinness, D. L., & Harmelen, F. (2013). OWL Web Ontology Language Overview, W3C Recommendation 10. Retrieved February, 2004, from http://www.w3.org/TR/2004/REC-owlfeatures-20040210/

[15] Ren, Y., *et al*. (2012). Rules and implementation for generating ontology from relational database. *Proceedings of the 2012 Second International Conference on Cloud and Green Computing (CGC)*.

[16] Kayed, A., Mohammad, N., & Mohammed, A. (2010). Ontology concepts for requirements engineering process in e-government applications. *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services*.

[17] Tim, B. L., James, H., & Ora, L. (2001). The semantic web. *Scientific American*, *284*, 34-43.

[18] W3C, OWL Working Group. Web Ontology Language (OWL). Retrieved, from http://www.w3.org/2004/OWL, 2004.

[19] Naïma, S. O., & Hafida, B. Y. A. (2015). A new OWL2 based approach for relational database description.

Information Technology and Computer Science, *1*.

[20] W3C Working Group. Defining N-ary Relations on the Semantic Web. Retrieved April 2006. from http://www.w3.org/TR/swbp-n-aryRelations/

Abdeljalil Boumlik was born in 1989, in Marrakech, Morocco. He is PhD student in the Department of Mathematics and Computer Science, Faculty of Science and Technology, University Hassan I, Settat, Morocco. His area of interest includes semantic web, web ontologies.

Mohamed Bahaj is a full professor in the Department of Mathematics and Computer Sciences from the University Hassan 1st Faculty of Sciences & Technology Settat Morocco. He is co-chairs of IC2INT, International Conference on Software Engineering, Databases and Expert Systems (SEDEXS'12) , NASCASE'11. He has published over 80 peer-reviewed papers. His research interests are intelligents systems, ontologies engineering, partial and differential equations, numerical analysis and scientific computing. He is associate editor of Journal of Artificial Intelligence and Journal Software Engineering.