Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USING FRAMEWORK TO SYNCHRONIZE ONTOLOGY WITH RELATIONAL DATABASE

[1]AHMED EL ORCHE, [2]MOHAMED BAHAJ
Faculty of Sciences & Technologies / Department of Computer and Information Science, Settat, Morocco
E-mail: [1]ahmed.elorche@gmail.com, [2]mohamedbahaj@gmail.com

ABSTRACT

Ontology is building bloc of semantic web. Ontologies evolve over time and not static. The synchronization of the relational database with the ontology is required once the ontology is changed. In this paper the authors propose an approach to synchronize the relational database with the ontology in two steps, firstly to match the relational database schema with the ontology, secondly to make design of a framework able to compare the current ontology's version with the pervious one, then manipulate the changes happened to build SQL queries to be executed in the relational database.

Keywords: *Synchronization, Semantic, Database, Datatype, Object Properties, OBDB, Ontology.*

1. INTRODUCTION

The web semantic was designed to display information in human-readable way. Thus, information access and information search will be more precise and more complete. The ontology is the building bloc of the web semantic. Database models, notably relational databases, have been the leader in last decades, enabling to store modify and extract information. The other side, ontologies have seemed as an alternative to databases in applications that require a semantic meaning.

The synchronization between the OBDB (ontology based database) and RDB is more important for some reasons as performance and development of the web semantic. That need every time launching the operation of update the ontology or the RDB or both of them. Many papers have begun the migration from RDB to ontology and in the reverse direction. The communication between ontologies and databases can be established if information represented by ontologies corresponds to data described in a database in a certain way.

In order to establish this communication the reference [8] pointed the following classification:

- Using the same conceptual modeling technique for representing ontologies and databases.
- Generating database schemas from ontologies.
- Obtaining Ontologies from database representations.
- Using OBDB (Ontologies Based Databases)

It means that always run a migration for the whole ontology or RDB and sometimes for a small change causing the impact of the performance of an entire system, for that we have to find solution to target just the changed objects. The reference [5] allows data updates specified as triplets to be propagated back to the relational database as tuples. Algorithms to translate the triplets to be updated, inserted, deleted into equivalent relational attributes/tuples whenever possible are presented. The reference [2] presents an alternative approach to managing RDF data in the database trough introducing a new Oracle object type for storing RDF data. The object type is built on top of the Oracle Spatial Network Data Model, which is Oracle's network solution in the database. This exposes the NDM functionality to RDF data, allowing RDF data to be managed as objects and analyzed as networks. The reference [3] presents a method for semantic and direct conversion of RDB to an ontology done in two steps, the first interested in schema and the second focuses on data. The reference [3] defined the model of ontology as a set of classes, a set of properties of datatype, a set of object properties, a list of individuals.

In Section 2 we clarify the related work of the paper. Section 3 introduces more the proposed approach. Section 4 describes the structure of the OBDB. Section 5 describes and gives algorithms and methods used by the framework proposed in this approach. Section 6 concludes the paper.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. RELATED WORK

The reference [1] proposes an approach to detect any changes applied in the ontology then generate SQL queries to synchronize the RDB with the OBDB. But the representation of columns length in the references [1] and [3] is missing. The representation of foreign key in the ontology in the references [3] makes the synchronization is irreversible from OBDB to RDB. The ability to migrate columns having the same name and belongs to different tables is very likely, that causing a problem of ambiguity in OBDB. In order to overcome these problems, we demonstrate in this paper a simple approach based on defining and make correspondence between the objects of database and owl classes, objects properties or datatypes properties of the ontology in a way to avoid all the problems already discussed above. In the same paper [1], the approach is based on a set of rules that generate SQL queries on a comparison of two versions of the ontology through SPARQL queries. The lack of the approach is the exceptions appear at the time of update of RDB. To avoid this problem, the current paper proposes new approach to update the RDB schema using the framework, this operation is based on the analysis and treatment of the extracted triplets of OBDB, then generate SQL script to be executed one shot in the RDB without errors.

3. PROPOSED APPROACH

The objective of this approach is to make a matching of the OBDB with the database. An update on OBDB requires an update in database. We will define as objects the columns, the constraints, the indexes and the tables of database then make correspondence between each one with classes in the OBDB. The link between a column and a table will correspond in ontology by object properties as well as between constraints/indexes and a column of database, the values of check constraints will be presented in the OBDB by datatype properties. Also the objective of this approach is how managing the triplets of ontology and how update the RDB regarding these triplets. Where does the idea of creating an intermediate framework able to store the triplets in specific tables, as well as it should follow algorithms for interoperability between SPARQL/SQL queries then periodically send SQL queries to the database if one or more changes in OBDB are happened.

4. ONTOLOGY BASED DATABASE

The reference [10] shows in the simplest case, an OWL class corresponds to a RDB table, an OWL datatype property corresponds to a table field, and an OWL object property corresponds to a foreign key.

In real life examples the mappings are not so straightforward. In this section we describe a direct translation of RDB to OBDB and demonstrate also the process of generation owl classes, owl datatype properties and object properties.

We assume that the OBDB will be generated from RDB. The structure of the OBDB schema adopted in this paper is:

OBDB = {{**C**}, {**DT**}, {**DO**}/
C is a set of classes corresponds to database objects,
DT is a set of datatypes corresponds to the objects type, or constraints values.
DO is a set of data objects corresponds to links between database objects
}

In this approach any object in database will correspond to a class in the OBDB, The tables and their columns will be linked by objects properties, the same goes for relations between indexes and columns, between constraints and columns and between constraints and constraints in the case of foreign keys.

As described in the figure 8, there is a single parent class of all others classes corresponds to the database objects: table, column, indexes and constraints. Among problems encountered in the modeling of the database constraints in OBDB are the difference between foreign key constraints and the others. That is why we propose to create class parent Constraint for each type of constraint classes (primary key, foreign key, unique, Not Null, check).

The presentation of foreign keys is different from a paper to another; in this paper we use two object properties to model this type of constraint. The first object property will be used to link the constraint with the column and the second will be used to link the constraint with the primary key of the referenced table.

Each table has one or more columns, obviously the names of these columns are different in the table, but another table may have a column with the same name, which an ambiguity in the modeling of OBDB appears. To find a solution to overcome this issue we propose to model the column by unique

Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

identifier instead the column name only as follow: table_name.colomn_name.

Example:

<column rdf:ID="**card.card_number**">

The datatype properties are used to define the characteristics of each object of ontology such as the name or type, but in this paper we use the datatype properties to model the value of a constraint if needed like the case of CHECK constraints, or to model the type and the length of columns.

The figure 1 shows the owl ontology classes that correspond to OBDB objects in Figure 9.

```
<owl:Class rdf:ID="object" />

<owl:Class rdf:ID="table">
    <rdfs:subClassOf rdf:resource="#object" />
</owl:Class>

<owl:Class rdf:ID="column">
    <rdfs:subClassOf rdf:resource="#object" />
</owl:Class>

<owl:Class rdf:ID="index">
    <rdfs:subClassOf rdf:resource="#object" />
</owl:Class>

<owl:Class rdf:ID="constraint">
    <rdfs:subClassOf rdf:resource="#object" />
</owl:Class>

<owl:Class rdf:ID="primaryKey">
    <rdfs:subClassOf rdf:resource="#constraint" />
</owl:Class>

<owl:Class rdf:ID="foreignKey">
    <rdfs:subClassOf rdf:resource="#constraint" />
</owl:Class>

<owl:Class rdf:ID="check">
    <rdfs:subClassOf rdf:resource="#constraint" />
</owl:Class>

<owl:Class rdf:ID="unique">
    <rdfs:subClassOf rdf:resource="#constraint" />
</owl:Class>

<owl:Class rdf:ID="notNull">
    <rdfs:subClassOf rdf:resource="#constraint" />
</owl:Class>
```

Figure 1: Owl ontology – 1

The figure 2 shows a part of owl ontology used to create object properties to link column with table.

```
<owl:ObjectProperty rdf:ID="hasColumn">
    <rdfs:domain rdf:resource="#table" />
    <rdfs:range rdf:resource="#column" />
</owl:ObjectProperty>
```

Figure 2: Owl ontology – 2

The figure 3 shows an example of three tables of RDB linked between them through foreign keys constraints. In the rest of this section we will describe how to model these tables with owl ontology.

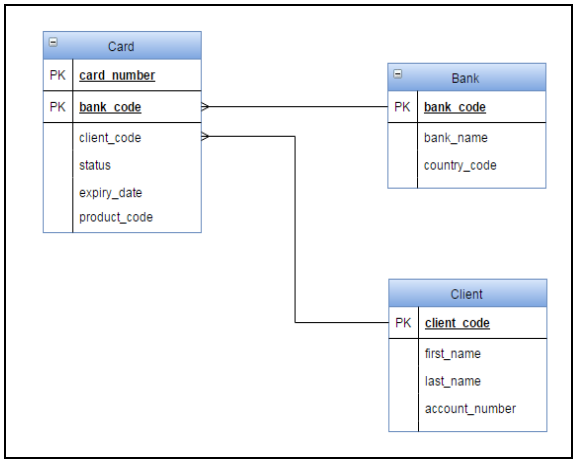

Figure 3: RDB tables

The figure 4 shows the part of the owl ontology generated to model the table "CARD" in the figure 3.

```
<column rdf:ID="bank.bank_code">
    <hasPrimaryKey rdf:resource="#pk_bank" />
    <datatype rdf:resource="CHAR(6)" />
</column>

<column rdf:ID="card.card_number">
    <hasPrimaryKey rdf:resource="#pk_card" />
</column>

<column rdf:ID="card.bank_code">
    <hasForeignKey rdf:resource="#fk_card_01" />
</column>

<column rdf:ID="card.status">
    <checkValue rdf:resource="#ch_card_status" />
</column>

<table rdf:ID="card">
    <hasColumn rdf:resource="#card.card_number" />
    <hasColumn rdf:resource="#card.bank_code"/>
    <hasColumn rdf:resource="#card.client_code"/>
    <hasColumn rdf:resource="#card.status"/>
    <hasColumn rdf:resource="#card.expiry_date"/>
    <hasColumn rdf:resource="#card.product_code"/>
</table>
```

Figure 4: Graph of data model - 2

The figure 5 shows a part of owl ontology used to create object properties, datatype properties to link columns with constraints.

Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
<owl:ObjectProperty rdf:ID="hasColumn">
    <rdfs:domain rdf:resource="#table" />
    <rdfs:range rdf:resource="#column" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasPrimaryKey">
    <rdfs:domain rdf:resource="#column" />
    <rdfs:range rdf:resource="#primaryKey" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasForeignKey">
    <rdfs:domain rdf:resource="#column" />
    <rdfs:range rdf:resource="#foreignKey" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasReferencedCol">
    <rdfs:domain rdf:resource="#foreignKey" />
    <rdfs:range rdf:resource="#primaryKey" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasCheck">
    <rdfs:domain rdf:resource="#column" />
    <rdfs:range rdf:resource="#check" />
</owl:ObjectProperty>

<owl:DataTypeProperty rdf:ID="checkValue">
    <rdfs:domain rdf:resource="#check" />
    <rdfs:range rdf:resource="xsd;string" />
</owl:DataTypeProperty>

<foreignKey rdf:ID="fk_card_01">
    <fkReferencedCons rdf:resource="#pk_bank" />
</foreignKey>

<check rdf:ID="ch_card_status">
    <checkValue rdf:resource="status in('N','S','R','B')" />
</check>

<primaryKey rdf:ID="pk_card" />
<primaryKey rdf:ID="pk_bank" />
```

Figure 5: RDB schema

5. FRAMEWORK

The framework is composed by its own database and programs to manage all triplets of OBDB; it should be supported as interface between OBDB and the RDB. Its database will contain different types of objects such as tables, primary keys, foreign keys, sequences, indexes and triggers. The figure 10 shows list of data model triplets extracted from the OBDB to be stored and managed by the framework.

The figure 6 describes the purpose proposed by the approach to ensure the synchronization from OBDB to RDB using the triplets of the ontology. The first step concerns the extraction of OBDB triplets by the framework using SPARQL queries, the purpose of the second step is how analyze and store data of ontology in the framework database, and in the third step after compare the obtained triplets with the triplets already stored, if there is change happened the framework database the program will generate SQL queries to be executed in RDB according to the change detected.

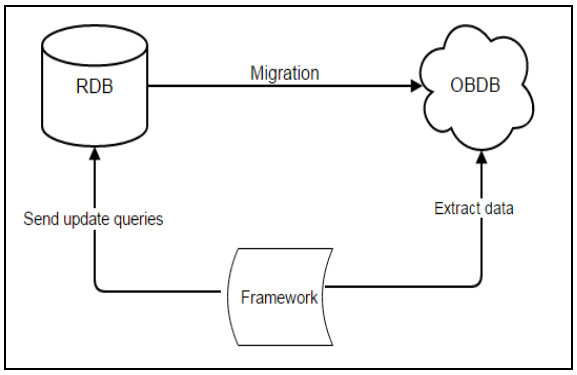

Figure 6: Ontology/RDB synchronization steps

The framework contains a set of tables created for the purpose to store the triplets of ontology. Register triplets in a single table is not practical for the synchronization task. That is why we adopt in this approach a set of tables. First type of tables is to keep the triplets under the form (subject, predicate, object). The second type of tables is to classify the triplets according to their different types of RDB objects (columns, tables, constraints, indexes), the third type of tables is to store the links between different objects or values of check constraints.

5.1 Framework Database Description

Table 1: Framework Database description

Table	Description
odb_triplets_df	Contains the old state of triplets of ontology
odb_triplets	Contains the current state of triplets of ontology
odb_tables	Contains RDB / RDB table name list
odb_columns	Contains the list of all the columns that exist in the tables of OBDB / RDB
odb_constraints	Contains all types of constraints of OBDB/RDB
odb_indexes	Contains all indexes of OBDB/RDB
odb_object_properties	Contains all objects properties to link columns with its tables
odb_datatypes	contient les objects properties pour lier les indexes et les colonnes de RDB

The Figure 11 describes the diagram data model of the framework database.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5.2 OBDB Interface

The ontology in the reference [1] is defined as four-tuple:

O = < C, DT, OP,I > Where:
C = {C_1, C_2 … C_n} is a set of classes
DT = {DT_1, DT_2 … DT_m} is a set of data type properties.
OP = {OP_1, OP_2 … OP_p} is a set of object properties.
I = {I_1, I_2 … I_p} is a set of individuals.

The current paper presents how to build SQL queries directly from an ontology through a set of rules and algorithms, the main idea is how to compare and detect changes between two versions of the same OBDB schema Oi and Oj where:

O_i = < C_i, DT_i, OP_i >
O_j = < C_j, DT_j, OP_j >

Now, we will adopt the approach rules used in the reference [1] to compare two list versions of <C>, <DT> or <DO> to detect changes.

The idea of this step is to make the extraction of all triplets of the ontology into ODB_TRIPLETS table. First we should copy the data of ODB_TRIPLETS to ODB_TRIPLETS_DF, second overwrite the content of ODB_TRIPLETS by new data extracted from ontology. To detect whether there are a differences between the two versions of the ontology we should compare the two tables ODB_TRIPLETS and ODB_TRIPLETS_DF. Each difference detected will be translated as a modification applied on the ontology. We will distribute different triplets detected on others tables of the framework database. At this stage the framework will also determine the type of operation applied to the triplet (I: insert, U: update, D: delete).

The different class's types of ontology are: {Table, Column, Index, PrimaryKey, ForeignKey, Not Null, Check, Unique}.

The different type of object properites of ontology are: {hasColumn, hasIndex, hasPrimaryKey, hasForeignKey, hasNotNull, hasCheck, hasUnique, hasReferencedCol}.

The algorithm below shows the process followed to compare two versions of ontology, then populate or update framework database according to the changes detected.

DECLARE

Tab_i *is cursor of triplet where predicate='type', object='table' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Col_i *is cursor of triplet where predicate ='type', object='column' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Idx_i *is cursor of triplet where predicate ='type', object='index' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Pk_i *is cursor of triplet where predicate ='type', object='primaryKey' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Fk_i *is cursor of triplet where predicate ='type', object='foreignkey' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Chk_i *is cursor of triplet where predicate ='type', object='Check' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

U_i *is cursor of triplet where predicate ='type', object='Unique' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

NN_i *is cursor of triplet where predicate ='type', object='NotNull' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

DT_i *is cursor of triplet where predicate ='checkValue' or = 'dataType' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

Tab_d *is cursor of triplet where predicate ='type', object='table' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

Col_d *is cursor of triplet where predicate ='type', object='column' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

Idx_d *is cursor of triplet where predicate ='type', object='index' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Pk_d *is cursor of triplet where predicate ='type', object='primaryKey' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

Fk_d *is cursor of triplet where predicate ='type', object='foreignkey' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

Chk_d *is cursor of triplet where predicate ='type', object='Check' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

U_d *is cursor of triplet where predicate ='type', object='Unique' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

NN_d *is cursor of triplet where predicate ='type', object='NotNull' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

DT_d *is cursor of triplet where predicate ='checkValue' or = 'dataType' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hcol_i *is cursor of triplet where predicate ='hasColumn' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hIdx_i *is cursor of triplet where predicate ='hasIndex' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hPk_i *is cursor of triplet where predicate ='hasPrimarykey' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hFk_i *is cursor of triplet where predicate ='hasForeignkey' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hChk_i *is cursor of triplet where predicate ='hasCheck' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hU_i *is cursor of triplet where predicate ='hasUnique' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hNN_i *is cursor of triplet where predicate ='hasNotNull' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hRc_i *is cursor of triplet where predicate ='hasReferencedCol' and triplet in ODB_TRIPLETS and not in ODB_TRIPLETS_DF;*

hcol_d *is cursor of triplet where predicate ='hasColumn' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hIdx_d *is cursor of triplet where predicate ='hasIndex' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hPk_d *is cursor of triplet where predicate ='hasPrimarykey' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hFk_d *is cursor of triplet where predicate ='hasForeignkey' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hChk_d *is cursor of triplet where predicate='hasCheck' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hU_d *is cursor of triplet where predicate ='hasUnique' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hNN_d *is cursor of triplet where predicate ='hasNotNull' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

hRc_d *is cursor of triplet where predicate ='hasReferencedCol' and triplet in ODB_TRIPLETS_DF and not in ODB_TRIPLETS;*

BEGIN
foreach tab in ***Tab_i*** *do*
record.table_name ← tab.subject;
reocrd.processing_step ← '1';
insert record in ODB_TABLES;
end foreach;

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
foreach Col in Col_i do
   record.table_name ← Col.subject;
   record.processing_step ← 'I';
   insert record in ODB_COLUMNS;
end foreach;

foreach Idx in Idx_i do
   record.table_name ← Idx.subject;
   record.processing_step ← 'I';
   insert record in ODB_INDEXES;
end foreach;

foreach Pk in Pk_i do
   record.constraint_name ← Pk.subject;
   record.constraint_type ← 'PK';
   record.processing_step ← 'I';
   insert record in ODB_ CONSTRAINTS;
end foreach;

foreach Fk in Fk_i do
   record.constraint_name ← Fk.subject;
   record.constraint_type ← 'FK';
   record.processing_step ← 'I';
   insert record in ODB_ CONSTRAINTS;
end foreach;

foreach Chk in Chk_i do
   record.constraint_name ← Chk.subject;
   record.constraint_type ← 'CH';
   record.processing_step ← 'I';
   insert record in ODB_ CONSTRAINTS;
end foreach;

foreach U in U_i do
   record.constraint_name ← U.subject;
   record.constraint_type ← 'U';
   record.processing_step ← 'I';
   insert record in ODB_ CONSTRAINTS;
end foreach

foreach NN in NN_i do
   record.constraint_name ← NN.subject;
   record.constraint_type ← 'NN';
   record.processing_step ← 'I';
   insert record in ODB_ CONSTRAINTS;
end foreach

foreach DT in DT_i do
   record.DT_name ← DT.subject;
   record.type ← DT.predicate;
   record.value ← DT.object;
   record.processing_step ← 'I';
   insert record in ODB_ DATATYPE;
end foreach

foreach tab in Tab_d do
   record.table_name ← tab.subject;
   record.processing_step ← 'D';
   Update record in ODB_TABLES;
end foreach;

foreach Col in Col_d do
   record.table_name ← Col.subject;
   record.processing_step ← 'D';
   Update record in ODB_COLUMNS;
end foreach;

foreach Idx in Idx_d do
   record.table_name ← Idx.subject;
   record.processing_step ← 'D';
   Update record in ODB_INDEXES;
end foreach;

foreach Pk in Pk_d do
   record.constraint_name ← Pk.subject;
   record.processing_step ← 'D';
   Update record in ODB_ CONSTRAINTS;
end foreach;

foreach Fk in Fk_d do
   record.constraint_name ← Fk.subject;
   record.processing_step ← 'D';
   Update record in ODB_ CONSTRAINTS;
end foreach

foreach Chk in Chk_d do
   record.constraint_name ← Chk.subject;
   record.processing_step ← 'D';
   Update record in ODB_ CONSTRAINTS;
end foreach;

foreach U in U_d do
   record.constraint_name ← U.subject;
   record.processing_step ← 'D';
   Update record in ODB_ CONSTRAINTS;
end foreach;

foreach NN in NN_d do
   record.constraint_name ← NN.subject;
   record.processing_step ← 'D';
   Update record in ODB_ CONSTRAINTS;
end foreach;

foreach DT in DT_d do
   record.DT_name ← DT.subject;
   record.type ← DT.predicate;
   record.value ← DT.object;
   record.processing_step ← 'D';
   insert record in ODB_ DATATYPE;
end foreach
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
  foreach hcol in hcol _i do
     record.linked_subject ← hcol.subject;
     record.link_type ← 'TAB';
     record.linked_object ← hcol.object;
     record.processing_step ← 'I';
     insert record in
ODB_OBJECT_PROPERTIES;
  end foreach;

  foreach hIdx in hIdx_i do
     record.linked_subject ← hIdx.subject;
     record.link_type ← 'IDX';
     record.linked_object ← hIdxobject;
     record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
 end foreach;

  foreach hPk in hPk_i do
     record.linked_subject ← hPk.subject;
     record.link_type ← 'PK'
     record.linked_object ← hPk.object;
     record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
 end foreach;

  foreach hFk in hFk_i do
     record.linked_subject ← hFk.subject;
     record.link_type ← 'FK'
     record.linked_object ← hFk.object;
     record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
 end foreach;

 foreach hU in hU_i do
    record.linked_subject ← hU.subject;
    record.link_type ← 'U'
    record.linked_object ← hU.object;
    record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
 end foreach;

foreach hNN in hNN_i do
    record.linked_subject ← hNN.subject;
    record.link_type ← 'NN'
    record.linked_object ← hNN.object;
    record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
 end foreach;

  foreach hRc in hRc_i do
    record.linked_subject ← hRc.subject;
    record.link_type ← 'R'
    record.linked_object ← hRc.object;
    record.processing_step ← 'I';
    insert record in ODB_OBJECT_PROPERTIES;
  end foreach;

  foreach hcol in hcol _d do
    record.linked_subject ← hcol.subject;
    record.link_type ← 'TAB'
    record.linked_object ← hcol.object;
    record.processing_step ← 'D';
    Update record in
ODB_OBJECT_PROPERTIES;
  end foreach;

foreach hIdx in hIdx_d do
   record.linked_subject ← hIdx.subject;
    record.link_type ← 'IDX'
    record.linked_object ← hIdx.object;
    record.processing_step ← 'D';
Update record in ODB_OBJECT_PROPERTIES;
 end foreach;

 foreach hPk in hPk_d do
    record.linked_subject ← hPk.subject;
    record.link_type ← 'PK'
    record.linked_object ← hPk.object;
    record.processing_step ← 'D';
Update record in ODB_OBJECT_PROPERTIES;
 end foreach;

 foreach hFk in hFk_d do
    record.linked_subject ← hFk.subject;
    record.link_type ← 'FK'
    record.linked_object ← hFk.object;
    record.processing_step ← 'D';
 Update record in ODB_OBJECT_PROPERTIES;
 end foreach;

 foreach hU in hU_d do
    record.linked_subject ← hU.subject;
    record.link_type ← 'U'
    record.linked_object ← hU.object;
    record.processing_step ← 'D';
Update record in ODB_OBJECT_PROPERTIES;
end foreach;

 foreach hNN in hNN_d do
    record.linked_subject ← hNN.subject;
    record.link_type ← 'NN'
    record.linked_object ← hNN.object;
    record.processing_step ← 'D';
Update record in ODB_OBJECT_PROPERTIES;
end foreach;

 foreach hRc in hRc_d do
    record.linked_subject ← hRc.subject;
    record.link_type ← 'R'
    record.linked_object ← hRc.object;
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

record.processing_step ← 'D';
Update record in ODB_OBJECT_PROPERTIES;
end foreach;
end;

5.3 RDB Interface

After updating the framework database, there remains only the update of RDB to complete the synchronization of the entire system. The issue encountered is that how to manage the SQL queries built by the framework.

All framework tables except ODB_TRIPLETS_DF contain the processing_step column to know if the record in the table is processed by the synchronization program or not yet. Each change occurred in the table is reflected at the column processing_step by the value 'X' to inform the program that the record was changed in the table. After the processing operation, the value of the column processing_step will be updated to 'Y'.

Using the table ODB OBJECT PROPERTIES, the program can gather all object of table (columns, indexes and contraints) and translate the binding of these objects together.

Example:

Build SQL queries to be executed in the RDB where there is one changed table in the OBDB.

The program will select the record of the table changed from the table **ODB_TABLE** where the column processing_step='X', then all columns, indexes and constraints will be selected from the table ODB_OBJECT_PROPERTIES. With the table ODB_DATATYPE, the program can select the types and the length of all columns. If the program detect modification in column object on the table ODB_COLUMNS, the program will select the name of the table where the column is belong through the table ODB_OBJECT_PROPERTIES, then begin the process of building the SQL queries like diagram in the figure 7 shows. The same diagram describes all cases and steps should be followed to ensure the update of RDB.

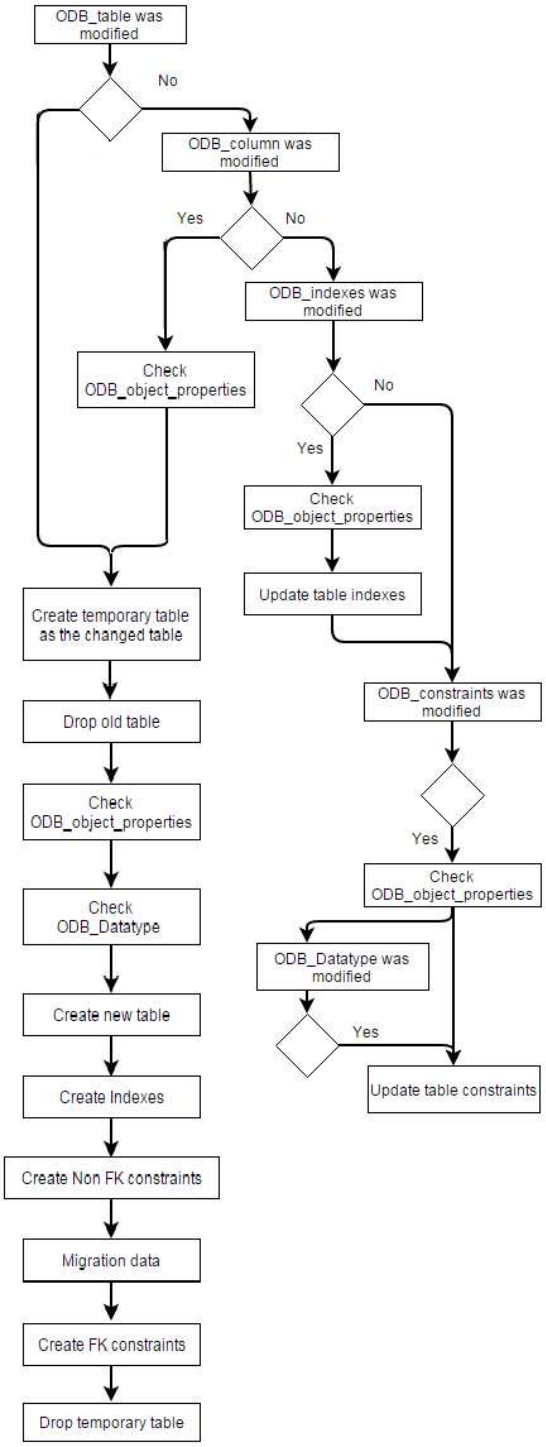

Figure 7: RDB Synchronization process

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6. CONCLUSION

In this paper, we show that our matching approach of RDB schema with OBDB keeps certain semantic characteristics of the RDB, reduce ambiguity problems causing obstacles in the case of recovering the ontology data. We also set up a design of a framework to ensure synchronization between RDB schema and OBDB on any change of ontology. The study of synchronization includes also the data of RDB is missing in this approach. the next future work will be focused on this goal.

REFERENCES:

[1] A. El Orche, M. Bahaj "A Method for Updating RDB of Ontology while keeping the Synchronization between the OWL and RDB", *ARPN Journal of Systems and Software*, Vol. 4, No4 July 2014, pp. 101-107.

[2] N. Alexander, S. Ravada "RDF Object Type and Reification in Oracle", *Oracle Corporation 1 Oracle Drive Nashua,* NH 03062 USA

[3] J. Bakkas, M. Bahaj, A. Marzouk "Direct Migration Method of RDB to Ontology while Keeping Semantics", *International Journal of Computer Applications (0975 – 8887),* Volume 65– No.3, March 2013.

[4] "Relationalizing RDF Stores for Tools Reusability" *DOI: 10.1145/1526709.1526855 Conference: Proceedings of the 18th International Conference on World Wide Web*, WWW 2009, Madrid, Spain, April 20-24, 2009

[5] Sunitha Ramanujam, Vaibhav Khadilkar, Latifur Khan, Steven Seida, Murat Kantarcioglu and Bhavani Thuraisingham "Bi-directional Translation of Relational Data into Virtual RDF Stores", *IEEE Fourth International Conference on Semantic Computing,* 2010.

[6] Ramanujam, S. "R2D: A Bridge between the Semantic Web and Relational Visualization Tools", *IEEE International Conference on Semantic Computing*, 2009, pp. 303-311.

[7] Angelina A. Tzacheva, Tyrone S. Toland, Peyton H. Poole, Daniel J. Barnes "Ontology Database System and Triggers" *Advances in Intelligent Data Analysis* XII Volume 8207 of the series Lecture Notes in Computer Science pp 416-426

[8] Carmen Martinez-Cruz, Ignacio J. Blanco, M. Amparo Vila "Ontologies versus relational databases: are they so different? A comparison" *Artificial Intelligence Review* December 2012, Volume 38, Issue 4, pp 271-290

[9] Anuradha Gali , Cindy X. Chen , Kajal T. Claypoo1 and Rosario Uceda-Sosa, "From Ontology to Relational Databases", http://www.cs.uml.edu/~cchen/ontology/comwim_04.pdf

[10] Guntars Bumans "Mapping between Relational Databases and OWL Ontologies: an Example" *Scientific Papers, University of Latvia*, 2010. Vol. 756 Computer Science and Information Technologies

[11] Vishal Jain, Dr. S. V. A. V. Prasad "Mapping Between RDBMS And Ontology: A Review" *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH,* VOLUME 3, ISSUE 11, NOVEMBER 2014 ISSN 2277-8

Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ANNEXURE

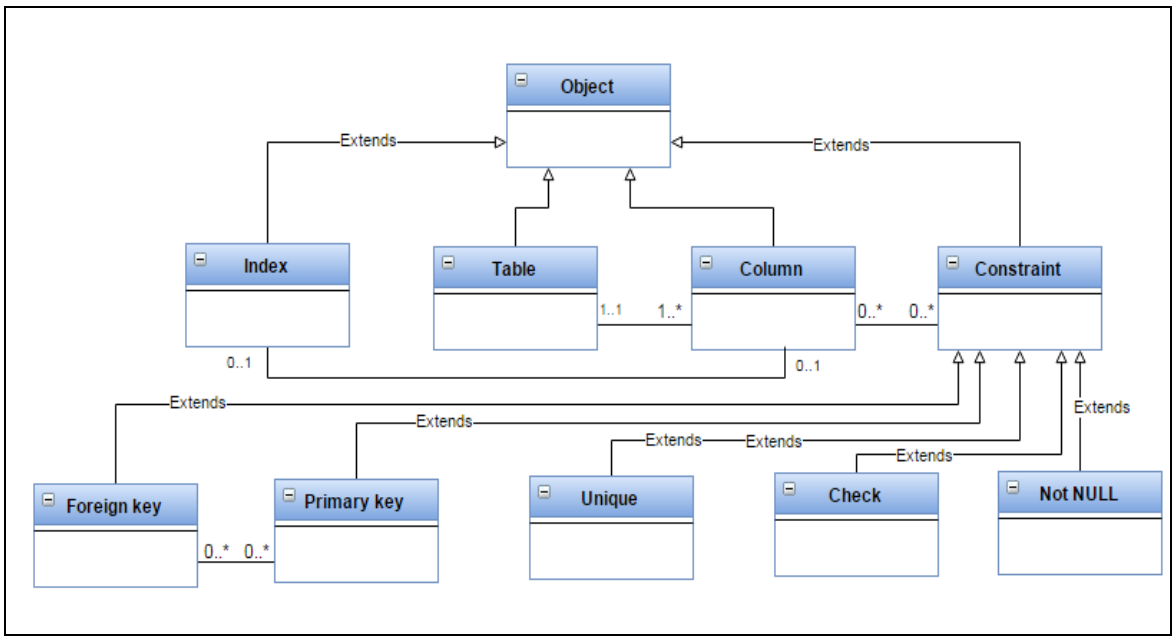

Figure 8: RDB Ontology Structure

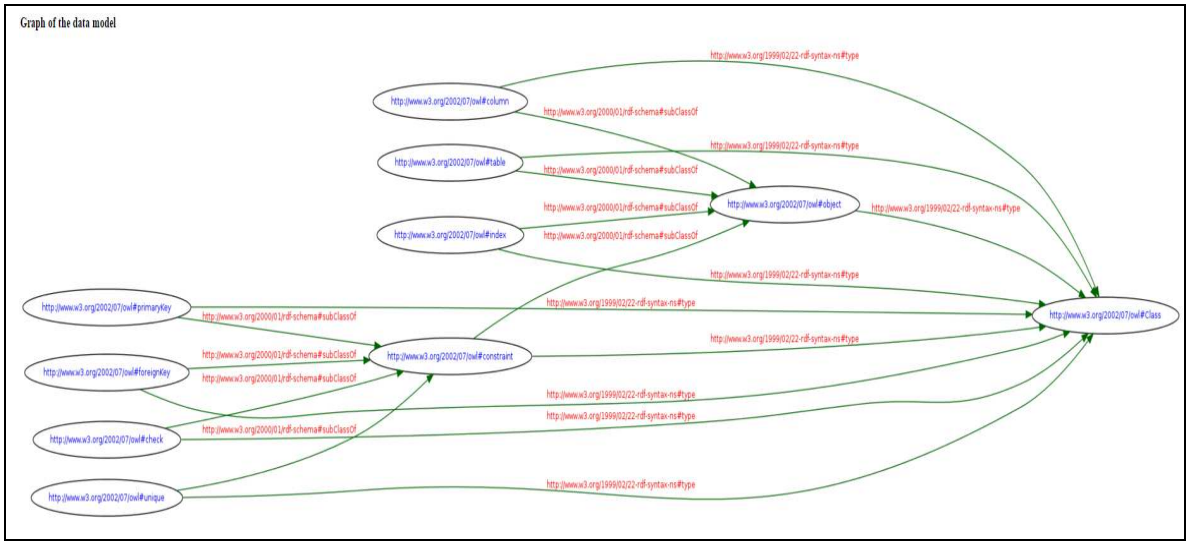

Figure 9: Graph Of Data Model - 1

Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://www.w3.org/2002/07/owl#object	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
2	http://www.w3.org/2002/07/owl#table	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
3	http://www.w3.org/2002/07/owl#table	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#object
4	http://www.w3.org/2002/07/owl#column	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
5	http://www.w3.org/2002/07/owl#column	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#object
6	http://www.w3.org/2002/07/owl#constraint	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
7	http://www.w3.org/2002/07/owl#constraint	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#object
8	http://www.w3.org/2002/07/owl#index	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
9	http://www.w3.org/2002/07/owl#index	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#object
10	http://www.w3.org/2002/07/owl#length	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
11	http://www.w3.org/2002/07/owl#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
12	http://www.w3.org/2002/07/owl#has_length	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
13	http://www.w3.org/2002/07/owl#has_length	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2002/07/owl#length
14	http://www.w3.org/2002/07/owl#has_length	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2002/07/owl#column
15	http://www.w3.org/2002/07/owl#has_type	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
16	http://www.w3.org/2002/07/owl#has_type	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2002/07/owl#type
17	http://www.w3.org/2002/07/owl#has_type	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2002/07/owl#column
18	http://www.w3.org/2002/07/owl#has_column	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
19	http://www.w3.org/2002/07/owl#has_column	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2002/07/owl#table
20	http://www.w3.org/2002/07/owl#has_column	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2002/07/owl#column
21	http://www.w3.org/2002/07/owl#has_index	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
22	http://www.w3.org/2002/07/owl#has_index	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2002/07/owl#table
23	http://www.w3.org/2002/07/owl#has_index	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2002/07/owl#index

Figure 10: Data Model Triplets

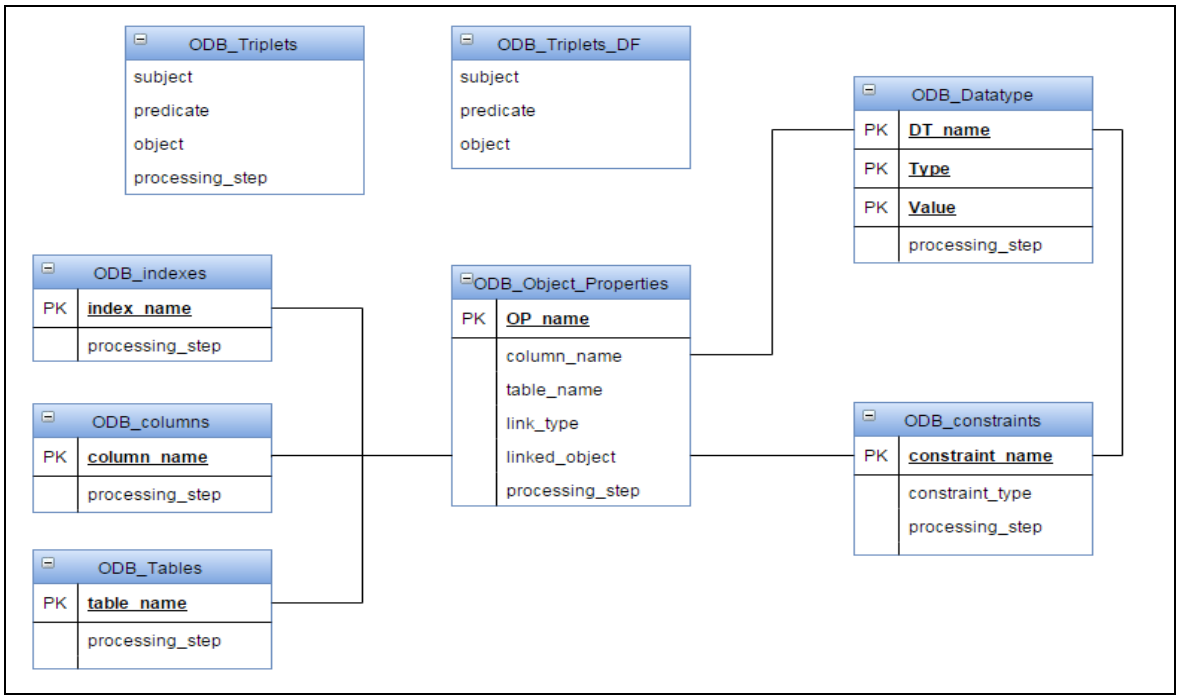

Figure 11: Framework Database