From Ontology to Relational Databases

Anuradha Gali[1], Cindy X. Chen[1], Kajal T. Claypool[1] and Rosario Uceda-Sosa[2]

[1] Department of Computer Science, University of Massachusetts, Lowell, MA 01854, U.S.A.
{agali | cchen | kajal}@cs.uml.edu
[2] IBM T. J. Watson Research Center, Hawthorne, NY 10532, U.S.A.
rosariou@us.ibm.com

Abstract. The semantic web envisions a World Wide Web in which data is described with rich semantics and applications can pose complex queries. Ontologies, a cornerstone of the semantic web, have gained wide popularity as a model of information in a given domain that can be used for many purposes, including enterprise integration, database design, information retrieval and information interchange on the World Wide Web. Much of the current focus on ontologies has been on the development of languages such as DAML+OIL and OWL that enable the creation of ontologies and provide extensive semantics for Web data, and on answering intensional queries, that is, queries about the structure of an ontology. However, it is almost certain that the many of the semantic web queries will be extensional and to flourish, the semantic web will need to accommodate the huge amounts of existing data that is described by the ontologies and the applications that operate on them. Given the established record of relational databases to store and query large amounts of data, in this paper we present a set of techniques to provide a lossless mapping of an OWL ontology to a relational schema and the corresponding instances to data. We present preliminary experiments that compare the efficiency of the mapping techniques in terms of query performance.

1 Introduction

The Semantic Web brings closer to realization the possibility of semantically organized data repositories, or *ontologies*, throughout the Internet that can be used for intelligent information searching both by humans and computational agents. Ontologies are human readable, comprehensive, sharable and formal which means that they are expressed in a language that has well-defined semantics. Ontologies are important to application integration solutions because they provide a shared and common understanding of data that exist within an application integration problem domain. Ontologies also facilitate communication between people and information systems. However, while today there is an un-precedented wealth of information available on the Web, to fully realize the power of ontologies and to enable *efficient* and *flexible* information gathering, persistent storage of ontologies and its subsequent retrieval is of paramount importance.

There are three possible approaches to store the ontology data and execute queries on that data. One is to build a special purpose database system which will be tailored to store and retrieve ontology specific data. The second is to use an object oriented database system exploiting the rich data modeling capabilities of OODBMS. Given the popularity of SQL and the efficiency of executing these queries, a complimentary approach is to use relational database systems. In this approach the OWL data is mapped into tables of a relational schema and the queries posed in an ontology language are translated into SQL queries. Relational database systems are very mature and scale very well, and they have the additional advantage that in a relational database, ontology data and the traditional structured data can co-exist making it possible to build applications that involve both kinds of data. While there are approaches that focus on translating XML documents into corresponding relational tables and data [9, 3], to the best of our knowledge ours is the first to focus on the persistent relational storage of OWL ontologies.

The rest of the paper is organized as follows: Section 2 gives an overview of OWL. Section 3 describes briefly the related work in terms of the different mapping schemes available for storing the XML documents in the relational database. The algorithm for translating the given OWL file into relational schema is explained in Section 4. Section 5 describes the preliminary performance comparison of the different approaches. We conclude in Section 6.

2 Background: Ontologies and OWL

Ontology is the key enabling technology for the semantic web. The main purpose of an ontology is to enable communication between computer systems in a way that is independent of the individual system technologies, information architectures and application domain. The key ingredients that make up an ontology are a vocabulary of basic terms, semantic interconnections, simple rules of inference and some logic for a particular topic.

Ontologies may vary not only in their content, but also in their structure, implementation, level of description, conceptual scope, instantiation and language specification. Ontologies are not all built the same way. A number of possible languages can be used, including general logic programming languages such as Prolog. More common, however, are languages that have evolved specifically to support ontology construction. The Open Knowledge Base Connectivity (OKBC) model and languages like KIF (and its emerging successor CL – Common Logic) are examples that have become the bases of other ontology languages. There are also several languages based on a form of logic thought to be especially computable, known as description logics. These include Loom and DAML+OIL, which are currently being evolved into the Web Ontology Language (OWL) standard. When comparing ontology languages, what is given up for computability and simplicity is usually language expressiveness, which many developers disagree since they feel that a good language should have a sound

underlying semantics.[1] gives a good overview of all the relevant technologies in Ontology.

OWL. OWL [6] is a language for defining and instantiating Web ontologies. An OWL ontology may include descriptions of classes, properties and their instances. Given an ontology, the OWL formal semantics specify how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms. The OWL language provides three increasingly expressive sub-languages designed for use by specific communities of implementers and users.

OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of `0` or `1`. It is typically simpler to provide tool support for OWL Lite than its more expressive relatives, and provide a quick migration path for thesauri and other taxonomies.

OWL DL supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a owl:DatatypeProperty can be marked as an owl:InverseFunctionalProperty. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.

3 Related Work - XML Mapping Approaches

Given the similarities between XML and OWL, in this section we give a brief overview of the mapping approaches that have been proposed in literature for XML documents.

Edge Approach. The simplest scheme proposed by Florescu et al. [3] is to store all attributes in a single table called the *Edge* table. The Edge table records the object identities (oids) of the source and target objects of the attribute, the name of the attribute, a flag that indicates whether the attribute is an inter-object reference or points to a value, and an ordinal number used to recover all attributes of an object in the right order and to carry out updates if the object

has several attributes with the same name. The Edge table, therefore, has the following structure:

```
Edge(source, ordinal, name, flag, target)
```

The key of the Edge table is `source, ordinal`. Figure 2 shows how the Edge table would be populated for the example XML graph.

Attribute Approach. Attribute mapping scheme groups all attributes with the same name into one table. Conceptually, this approach corresponds to a horizontal partitioning of the Edge table used in the first approach, using name as the partitioning attribute. Thus, as many Attribute tables are created as for different attribute names in the XML document, and each Attribute table has the following structure:

```
Aname(source, ordinal, flag, target)
```

The key of the Attribute table is `source, ordinal`, and all the fields have the same meaning as in the Edge approach.

Universal Approach. Universal table approach stores all the attributes of an XML document. This corresponds to a Universal table with separate columns for all the attribute names that occur in the XML document. Conception ally, this Universal table corresponds to the result of an outer join of all Attribute tables. The structure of the Universal table is as follows, if n1, n2,...,nk are the attribute names in the XML document, then

```
Universal(source, ordinaln1, flagn1, targetn1, ordinaln2,
          flagn2, targetn2, . . . , ordinalnk, flagnk, targetnk)
```

The Universal table has many fields which are set to null, and it also has a great deal of redundancy.

Normalized Universal Approach. The UnivNorm approach, is a variant of the Universal table approach. The difference between the UnivNorm and Universal approach is that multi-valued attributes are stored in separate Over-flow tables in the UnivNorm approach. For each attribute name that occurs in the XML document a separate Overflow table is established, following the principle of the Attribute approach. This way, there is only one row per XML object in the UnivNorm table and that table is normalized. The structure of the UnivNorm table and the Overflow tables is as follows, if n1; : : : ; nk are all the attribute names in the XML document, then

```
UnivNorm(source, ordinaln1, flagn1, targetn1, ordinaln2, flagn2,
         targetn2, . . . , targetnk)
OverflowOwn1(source, ordinal, flag, target)
OverflowOwnk(source, ordinal, flag, target)
```

The key of the UnivNorm table is `source`. The key of the Overflow table is `fsource, ordinalg`.

Basic Inlining Approach. Basic Inlining Approach converts the XML DTD into relations. Basic inlines as many descendants of an element as possible into a single relation. Basic creates relations for every element because an XML document can be rooted at any element in a DTD. Basic approach is highly inefficient since it creates a large number of relations.

Shared Inlining Approach. Shared Inlining tries to avoid the drawbacks of basic by ensuring that an element node is represented in exactly one relation. Shared identifies the element nodes that are represented in multiple relations in Basic and shares them by creating separate relation for these elements. In shared, relations are created for all elements in the DTD graph whose nodes have an in-degree greater than one. Nodes with an in-degree of one are inlined. Element nodes having an in-degree of zero are also made into separate relations, because they are not reachable from any other node. Elements below a "*" node are made into separate relations.

Hybrid Inlining Approach. The hybrid inlining technique is the same as shared except that it inlines some elements that are not inlined in shared. Hybrid additionally inlines elements with in-degree greater than one that are not recursive or reached through a "*" node.

4 Mapping OWL to Relational Schema

Semantic web is increasingly gaining popularity and importance with an increasing number of people using ontologies to represent their information. To augment the growth of ontology, efficient persistent storage of ontology concepts and data is essential. We now examine relational databases as an effective persistent store for ontologies. While there is research on storing XML documents in relational databases, there has not been much work done on the persistent storage of ontologies, in particular of ontologies described by OWL.

In this section, we present an approach to map OWL to relational schema. The goal of our work is to devise an approach that will result in fewer joins when a user issues a query since join is an expensive operation. Since OWL is based on XML Syntax, but defines a lot of constraints on classes and provides inference, the approaches to store XML documents in relational databases are not fully applicable. We need to preserve all the constraint information while mapping OWL into relational schemas.

4.1 System Architecture

Our OWL to relational schema mapping system contains three parts — Ontology Modeler, Document Manager and Ontology Resource Manager.

`Ontology Modeler` takes OWL documents as input and creates an ontology model that is similar to DOM [2] for XML documents. While parsing the OWL documents, all the constraints will also be detected and recorded. An appropriate

OWL reasoner, depending upon the input document if its OWL Lite, OWL full or OWL DL, is selected. Currently, all the OWL documents are stored in main memory for parsing.

`Document Manager` helps in processing and handling of OWL documents. It uses Jena [4] to assist importing of OWL documents. This is important because if we do not do this, once the imports have been processed, it would be impossible to know where a statement came from. Document Manager helps in processing and handling multiple OWL documents. It builds the union of the imported documents and creates new models for the imported OWL documents.

`Ontology Reasoner` provides methods for listing, getting and setting the RDF [8] types of a resource. The `rdf:type` property defines many entailment rules in the semantic models of the various ontology languages.

4.2 OWL2DB Algorithm

The algorithm we developed to map an OWL document into relational table is given in Algorithm 1.

Algorithm 1 Mapping OWL to Relational Schema

1: The given OWL file is first parsed for Root classes that returns an iterator over all the root classes.
2: Iterate through the root classes to determine the depth of the descendants.
3: The showclass method is called that keeps track of the depth of the subclasses from the root and when the depth of the graph is 3 then a relation table is created for all the subclasses above it with attribute names/instances as column names.
4: The above step is repeated starting from the last subclass and traversing the graph and collecting the list of all the table names and column names of subclasses that are a the depth of three from the starting node.
5: The depth of the Union/complement/disjoint subclasses on the same level are the same.
6: Separate relations are created for classes that are disjoint from each other.
7: Once the information on tablenames and corresponding attributes are collected, we need to collect the values of the instances to populate the database. The graph is parsed again to collect the values that are stored in a data structure.
8: Finally, a database connection is established and all the tables are created and then the values are inserted. Inlining the root is handled the same way as a parent node.

Figure 1 shows an example OWL document. Table 1, 2, 3, 4 represent the relational tables into which the OWL document was mapped.

```
<rdf:RDF xmlns:xsi="http://www.w3.org/2001/XMLSchema-
          instance" xsi:noNamespaceSchemaLocation=
  "C:\usr\Repository\iris1.0\data\schema\IRISData.xsd"
           Name="FurnitureOntology">
<owl:Ontology rdf:about="">
```

```
<rdfs:comment>An example of OWL Ontology</rdfs:comment>
<owl:priorVersion rdf:resource="C:\usr\Repository\iris1.0
                \data\schema\IRISData.xsd"/>
<owl:imports rdf:resource="http://www.w3.org/2001/XMLSchema-instance"/>
<rdfs:label>Furnite Ontology</rdfs:label>

<!----Main Classes----->
<owl:Class rdf:ID="OntologyDependency"/>
   <rdfs:label xml:lang="en">ShoppingCatalogOntology</rdfs:label>
 </owl:Class>

<owl:Class rdf:ID="Relation"/>
  <rdfs:label>Relation/rdfs:label>
</owl:Class>
<owl:Class rdf:ID="ProtoNode"/>
  <rdfs:label>ProtoNode</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="AtomicNode"/>
  <rdfs:label>Atomic Node</rdfs:label>
</owl:Class>

 <!----Sub Classes----->
    <owl:Class rdf:ID="Name"/>
      <rdfs:subClassOf rdf:resource="#Relation" />
      <rdfs:label xml:lang="en">Name</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Arity"/>
      <rdfs:subClassOf rdf:resource="#Relation" />
      <rdfs:label xml:lang="en">Arity</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Reflexive"/>
      <rdfs:subClassOf rdf:resource="#Relation" />
      <rdfs:label xml:lang="en">Reflexive</rdfs:label>
    </owl:Class>
      <owl:Class rdf:ID="Name"/>
              <rdfs:label>Name</rdfs:label>
              <rdfs:subClassOf>
              <owl:Restriction>
                      <owl:onProperty rdf:resource="#ProtoNode"/>
                      <owl:DatatypeProperty rdf:ID="string">
              </owl:Restriction>
              </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:ID="IsAbstract"/>
              <rdfs:label>IsAbstract</rdfs:label>
```

```
            <rdfs:subClassOf>
            <owl:Restriction>
                    <owl:onProperty rdf:resource="#ProtoNode"/>
                    <owl:DatatypeProperty rdf:ID="string">
            </owl:Restriction>
            </rdfs:subClassOf>
 </owl:Class>
- <owl:Class rdf:ID="ProtoNode"/>
            <rdfs:subClassOf>
            <owl:Restriction>
                    <owl:onProperty rdf:resource="#ProtoNode"/>
            </owl:Restriction>
            </rdfs:subClassOf>
          </owl:Class>
            - <owl:Class rdf:ID="Name"/>
                    <rdfs:label>Name</rdfs:label>
                    <rdfs:subClassOf rdf:resource="#ProtoNode" />
                    <owl:Restriction>
                            <owl:DatatypeProperty rdf:ID="string">
                    </owl:Restriction>
              </owl:Class>
             - <owl:Class rdf:ID="constraintNexus"/>
                    <rdfs:label>Constraint Nexus</rdfs:label>
                    <rdfs:subClassOf>
                    <owl:Restriction>
                            <owl:onProperty rdf:resource="#ProtoNode"/>
                    </owl:Restriction>
                    </rdfs:subClassOf>
              </owl:Class>

                    - <owl:Class rdf:ID="Relation"/>
                            <rdfs:label>Relation</rdfs:label>
                            <rdfs:subClassOf rdf:
                                resource="#constraintNexus" />
                            <owl:Restriction>
                                    <owl:DatatypeProperty rdf:ID="string">
                                    <rdfs:domain rdf:resource=
                                            "#ProtoNode"/>
                            </owl:Restriction>
                      </owl:Class>
```

Fig. 1. Sample OWL document.

Name	*OntologyDependency*
Furniture Ontology	Shopping Cart

Table 1. Furniture Table

Name	*Arity*	*Reflexive*	*Symmetric*
SPATIAL RELATION	2	true	true
HAS-OWNER	2	false	false

Table 2. Relation Table

5 Performance

To test the performance of our system, we generated two OWL files which are wide and deeply nested based on `wine.owl` and `food.owl` [7]. The OWL file was parsed and separate relations were created for all children nodes at a depth of 2. The nodes with an in-degree less than 2 were inlined. The experiment was repeated, creating relations for child nodes that were at depth 3, 4, 5, ..., 10, etc. After each mapping, queries were executed to determine the depth-level at which the inlining should occur for optimal performance. Based on the result, we have decided that the optimal performance is obtained when the nodes with an in-degree of 3 to 5 are inlined based on the depth of the OWL document. When the inlining increases more than that then the performance is close to the performance of Universal Table approach.

Experimental evaluations were conducted on all the different mapping schemes discussed in Section 3 and also for the OWL to relations mapping. The database used for the performance evaluation is IBM DB2 [5]. The parameters considered were time taken and number of joins needed to perform a simple select or update query. Separate databases were created for each mapping schemes. The same set of Queries were issued on all the databases.and the results are shown in Figure 2, 3, 4, and 5.

The queries that were used are as follows.

Query 1 *A simple select query to display all the information in the table.*

Select hasColor,hasLocation, hasBody, hasSugar from Wine

ProtoName	*IsAbstract*	*childProtoName*
Shopping Catalog Base	true	Physical Object

Table 3. Proto Table

ChildProtoName	*IsAbstract*	*SemIndp*	*ConstraintRelation*	*ConstraintTarget*
Physical Object	true	false	IS-A	ShoppingCatalogBase

Table 4. Child Proto Table

Query 2 *A delete query.*

Delete hasSugar from all the Wines that have hasColor=Red

Query 3 *An update query.*

Update the hasBody for all the Wines located in Sauterine Region

Query 4 *A select query with joins.*

List all the Wines made in Bordeaux (several location)

The performance evaluation showed in Figures 2, 3, 4, 5 [3] clearly indicate that Basic Inlining, Universal and Normalized Universal approach perform worst among the other methods. It also proves that OWL2DB performs pretty close to shared and attribute mapping approaches that are known to be the optimal mapping approaches from XML to Relations. OWL2DB is therefore a clearly an optimal mapping from OWL to Relations.

6 Conclusion

Ontology has more expressive power than XML but XML is richer in defining structures and grammar. Ontology and XML can be used to describe semi-structured natural language texts. Ontology is about sharing and consensual terminology and XML is used to accomplish that. Ontology provides a domain theory while XML provides the structure of data container. Ontology is a powerful new tool for building enterprise and web-based information systems that hold the promise of drastically reducing system construction and maintenance costs. It helps businesses develop information systems that are reusable, maintainable, and able to talk with other systems sharing similar kinds of information. XML can be used as the underlying technology for Ontology since it is pretty mature. Both XML and Ontology together can pave the way for the dynamic knowledge management systems.

This paper describes an efficient approach to mapping OWL documents into relational tables. There has not been much work in this area and we have tried to make the process autonomous, that is there is no human intervention once the input document is given. As the semantic web is gaining importance lately, we need an efficient approach to store all the ontology documents for the ease of querying and this paper introduces a first attempt in this direction.

[3] In these figures, 1,2,3,4,5,6,7,8 of the X-Axis stands for the Edge, Attribute, Universal, UnivNorm, Basic, Shared, Hybrid, OWL2DB mapping schemes.

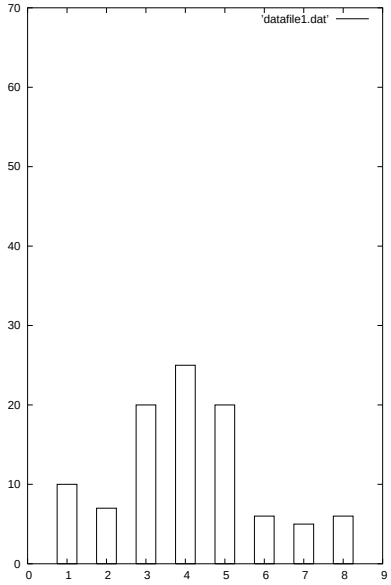

Fig. 2. Simple Select Query Q1

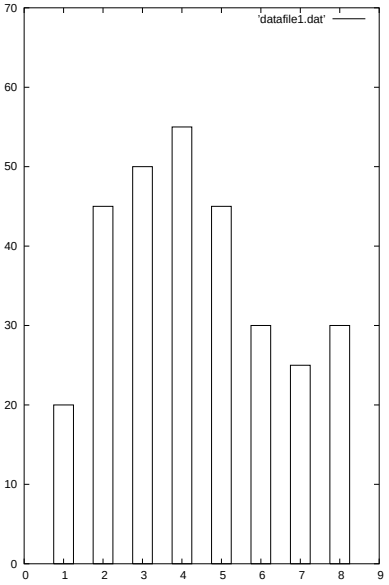

Fig. 3. Update Query Q2

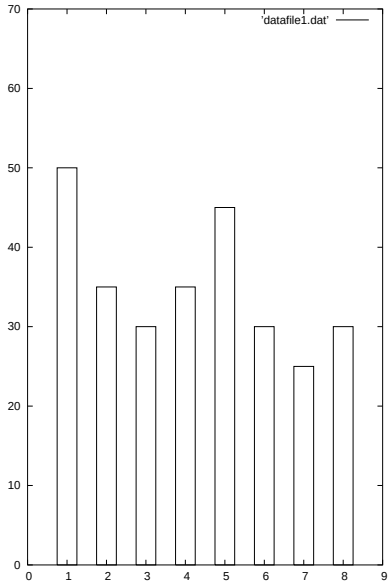

Fig. 4. Delete Query Q3

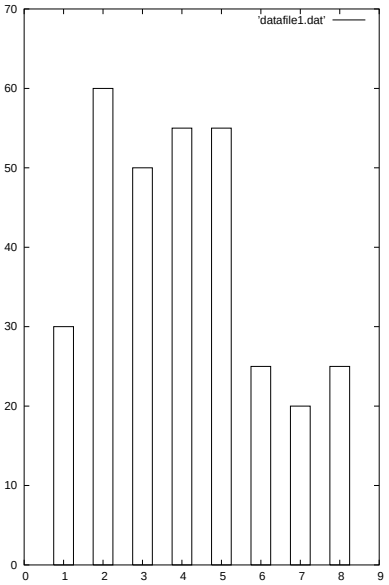

Fig. 5. Join Query Q4

References

1. John Davies, Dieter Feasel, and Frank Van Harmelen. *Towards the Semantic Web: Ontology Driven Knowledge Management*. John Wiley snd Sons Inc, 2003.
2. Document Object Model (DOM) Level 3 Core Specification. *http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/*.
3. Daniela Florescu and Donald Kossman. Performance evaluation of alternate mapping schemes for stoing xml data in a relational database. *Technique Report*, 1999.
4. Jena Documentation. *http://jena.sourceforge.net/ontology/index.html*.
5. David Martineau. *DB2 Universal Database for Application Development*. John Wiley snd Sons Inc, 1998.
6. OWL Web Ontology Language. *http://www.w3.org/TR/owl-guide/*.
7. OWL Web Ontology Language Reference. *http://www.w3c.org/TR/owl-ref/*.
8. RDF Primer. *http://www.w3c.org/TR/2004/REC-rdf-primer-20040210/*.
9. Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, and Jeffrey Naughton. Relational databases for querying xml documents: Limitations and opportunities. *Proceedings of 25 VLDB Conference*, 1999.