DLDB: Extending Relational Databases to Support Semantic Web Queries

Zhengxiang Pan
(Lehigh University, USA
zhp2@cse.lehigh.edu)

Jeff Heflin
(Lehigh University, USA
heflin@cse.lehigh.edu)

Abstract: We present DLDB, a knowledge base system that extends a relational database management system with additional capabilities for DAML+OIL inference. We discuss a number of database schemas that can be used to store RDF data and discuss the tradeoffs of each. Then we describe how we extend our design to support DAML+OIL entailments. The most significant aspect of our approach is the use of a description logic reasoner to precompute the subsumption hierarchy. We describe a lightweight implementation that makes use of a common RDBMS (MS Access) and the FaCT description logic reasoner. Surprisingly, this simple approach provides good results for extensional queries over a large set of DAML+OIL data that commits to a representative ontology of moderate complexity. As such, we expect such systems to be adequate for personal or small-business usage.

Keywords: DAML+OIL, Knowledge Base, Relational Database, Description Logic Reasoner, Storing RDF

1 Introduction

DAML+OIL enables the creation of ontologies and provides extensive semantics for Web data. This language is heavily influenced by description logics. Research on DL reasoners is primarily focused on intensional queries, that is, queries about the structure of an ontology. However, it is almost certain that the majority of Semantic Web queries will be extensional ones. Databases are excellent tools for storing and querying data, but lack the ability to perform the inference sanctioned by DAML+OIL entailments. This paper describes one method to extend relational databases to support DAML+OIL semantics.

Making use of the FaCT DL reasoner [Horrocks, 00], DLDB has been successfully implemented on a common RDBMS: Microsoft Access. It can process, store and query DAML+OIL formatted semantic content. The main purpose of this system is to investigate how DL reasoning and relational database systems can be combined to support extensional queries about DAML+OIL documents. By extensional, we mean queries that concern ground data, as opposed to queries about the structure of the ontologies. This system is optimized for ontologies of moderate sizes (at the magnitude of hundreds of classes and properties).

2 Design

Since DAML+OIL builds on RDF, we will first look at how to represent RDF information in a database. Then, we describe how to add support for RDF and DAML+OIL inference to our design.

2.1 RDF(S) Storage in Relational Databases

Although there are differences between RDF's graph-based model and the semi-structured model of XML, our work can benefit from the research on storing XML data in relational databases [Florescu and Kossman, 99].

Horizontal DB: Horizontal schema [Agrawal et al., 01] only need one "universal" table in the database. Every individual (instance) falls into one record in the table. While the data model is simple, there are some drawbacks within this approach: large number of columns; limits on property values; sparsity, etc.

Vertical table: In [Alexaki et al., 01], it is also named the "Generic Representation". This approach has a single table where each record corresponds to a RDF triple. However, this design means that any query has to search the whole database and queries that involve joins will be especially expensive.

Horizontal class: This approach is similar to the horizontal database approach but there is a separate table for each class in the ontology. This essentially corresponds to the entity-relational approach frequently used when designing databases.

Table per property: Yet another alternative is to assign a table to each property:

```
PROPERTY_name(Subject,Object)
```

In the database community, this approach is called the "decomposition storage model" [Agrawal et al., 01]. Like the vertical table approach, queries involving the implicit instances of a class can be particularly expensive.

Hybrid approach: We adopted an approach that combines the property table approach with the horizontal class aproach (see Figure 1). It is similar to the "Specific Representation" in [Alexaki et al., 01]. According to this model, creating tables corresponds to the definition of classes or properties in ontology. The classes or properties' ID should serve as the table names.

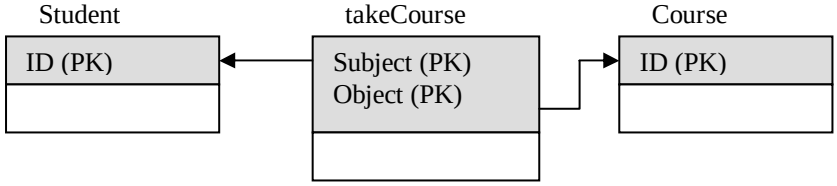

Figure 1: Example data model for a small ontology using Hybrid design

For ontologies with a moderate number of classes (a few hundred at most), this approach should perform well for simple queries. The experiments of [Agrawal et al., 01] showed that the table per property approach performed as well as or better than the horizontal and vertical approaches. However, ontologies with too many classes (e.g. the openDirectory ontology) either can't be accommodated in the underlying database or have a significant overhead [Alexaki et al., 01]. Yet, such schemas are rare, as discovered by the survey in [Magkanaraki et al., 02].

2.2 RDF(S) Entailment in Relational Databases

RDF entailment is relatively simple, mostly consisting of taxonomic inference using *rdfs:subClassOf* and *rdfs:type*, and similar inference for *rdfs:subPropertyOf*.

In our system class hierarchy information is stored through views. A view is a form of query in relational database. Some early research on loading data into DL reasoners also addressed similar method. In our design, the view of a class is defined recursively (see figure 2). It is the union of its table and all of its direct subclasses' views. Hence, a class's view contains the instances that are explicitly typed, as well as those that can be inferred.

In the terminology of deductive databases, the views are intensional database (IDB) relations which are defined by logical rules. The tables are extensional database (EDB) relations which store the explicit information from the documents.

For example, consider the following statements in a RDF model:

```
<rdfs:Class rdf:ID=″Student″/>
<rdfs:Class rdf:ID="UndergraduateStudent">
    <rdfs:subClassOf rdf:resource="#Student" />
<rdfs:Class/>
```

The Class view creation algorithm will define the view of Student as:

```
SELECT * FROM Student
UNION SELECT * FROM UndergraduateStudent_view;
```

function CreateViews(R)
  inputs: R, a RDF model
  static: $T_1, T_2, \ldots T_n$, a set of database tables
    $V_1, V_2, \ldots, V_n$, a set of database views

  for all triples (type x Class) $\in R$ do
    let T_x be the table containing explicit instances of class x
    let V_x be the view corresponding to class x
    $V_x \leftarrow T_x$
  for all triples (subClassOf y x) $\in R$ do
    let V_x be the view corresponding to class x
    if $\neg$ ($\exists x'$, ((subClassOf y x') $\in$ R) $\wedge$ ((subClassOf x' x) $\in R$)) then
      let V_y be the view corresponding to class y
      $V_x \leftarrow V_x \cup V_y$

Figure 2: RDF Class view creation algorithm

The *rdfs:subPropertyOf* relationship between properties is implemented in a similar way. The only remaining RDF entailments are those that entail class membership based on the use of a property and its domain or range. This could be accommodated in the view approach by adding another view to the union; however we do not currently implement this.

2.3 Supporting DAML+OIL Entailment

DAML+OIL has many features from description logics, the most significant are the constraints for class description. Using these, a DL reasoner can compute class

subsumption, i.e., the implicit subClassOf relations. Our database design can benefit from subsumption by using a DL reasoner to precompute subsumption, and then using the inferred class taxonomy to create our class views.

Using the above mechanism, our system stores the results of subsumption and only consults the DL reasoner once for each new ontology in the knowledge base. Whenever queries are issued concerning the instances of the ontology, the inferred hierarchy information can be automatically utilized. The intuition here is that ontologies don't change frequently although they can be imported or referred to many times. Thus, precomputation improves the computational efficiency which can save time as well as system resources. Note that at this time, we have not considered those elements (features) that are not related to subsumption. They are: *daml:inverseOf*, *daml:equivalentTo*, *daml:hasValue*, *daml:sameIndividualAs*, *daml:UnambigousPropery* and *daml:UniqueProperty*, etc.

3 Implementation

In our implementation, we use Microsoft Access as our relational database management system. In addition to our table design, some details should be taken into account when implementing the database schemas for the system. We use a 'ONTOLOGY-INDEX' table to manage loaded ontologies' infomation in the database. We include a 'source' field in each class and property table to support tracing the source document. In order to shrink the size of database and hence reduce the average query time, we assign each URI a unique ID number in our system. We use a indexed 'URI-INDEX' table to record the URI-ID pairs. We also use a hash table to cache every URI-ID pair found during the current loading process.

In our implementation, we borrow some code from OilEd [Bechhofer et al., 01] and use FaCT [Horrocks, 00] as our reasoner. First, a DAML parser borrowed from OilEd parses the original ontology source file to an ontology object, and then is translated into an equivalent SHIQ knowledge base and serialized to a temporary XML-formatted file. The reasoner running on the FaCT server reads that XML file to construct concepts, checks for the consistency of classes, determines the implicit subsumption relationships and reports what has been discovered by rewriting that temporary XML file. After the reasoner terminates, the program rearranges the class hierarchy in the ontology object based on the temporary XML file. DLDB then creates tables and views for corresponding classes and properties using a variation of the algorithm from Section 2.2.

The query API in DLDB currently supports conjunctive queries in KIF-like format and is implemented as a set of Java classes. During execution, each original query is translated into a standard SQL query sentence and sent to the database via JDBC. Then the database's DBMS processes the SQL query and returns appropriate results.

4 Conclusion

Preliminary experiments show that the use of views in a relational database and the FaCT reasoner make the results much more complete for some queries, while the costs (such as the increases on query time, loading time and database size) are considerably low or even negligible. Table 1 shows how the performance of DLDB

scales with the number of instances in the system. All these prove that the idea of using description logic to extend the relational database is feasible.

Although MS Access is not particularly scaleable, and would not be suitable for a large-scale knowledge portal, we see this system as fitting the needs of the personal or small business user who wishes to take advantage of semantic web technology. The integration of a common desktop database system with basic description logic reasoning gives such users the best of both worlds. It is also important to note that our design is not dependent on Access. We believe that given a suitable underlying RDBMS, this approach will scale well, and this is one of our chief directions for future work. Other future directions include adding support for more RDF and DAML+OIL entailments, and experimenting with the performance of various design alternatives.

Number of instances	Loadtime (hr:min:sec)	Size on disk (KB)	Typical query time (ms)
17,150	0:6:51	13,042	32 - 418
107,421	0:22:39	73,925	200 - 4,962
218,690	1:27:24	147,949	396 - 11,953
462,316	3:06:32	311,099	881 - 32,948
1,146,186	7:59:46	766,738	2295 - 115,782

Table 1: Performance of DLDB

Acknowledgement

Some of the material in this paper is based upon work supported by the Air Force Research Laboratory, Contract Number F30602-00-C-0188. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

Reference:

[Agrawal et al., 01] R. Agrawal, A. Somani, and Y. Xu. *Storage and Querying of E-Commerce Data.* In Proc. of VLDB 2001

[Alexaki et al., 01] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis & K.Tolle, *On Storing Voluminous RDF Description: The case of Web Portal Catalogs*, In Proc. of WebDB2001 in conjunction with ACM SIGMOD'01 Conference, 2001.

[Bechhofer et al., 01] S. Bechhofer, I. Horrocks, C. Goble and R. Stevens. *OilEd: a Reasonable Ontology Editor for the Semantic Web*. In Proceedings of KI2001, Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[Florescu and Kossman, 99] D. Florescu and D. Kossman. *A performance evaluation of alternative mapping schemes for storing XML data in a relational database*. Technical report, INRIA, France, May 1999.

[Horrocks, 00] I. Horrocks. *Benchmark Analysis with FaCT*. In Proc. TABLEAUX 2000, pages 62-66, 2000.

[Magkanaraki et al., 02] A. Magkanaraki, S. Alexaki, V. Christophides, and D. lexousakis. *Benchmarking RDF Schemas for the Semantic Web*. In Proceedings of the First International Semantic Web Conference (ISWC 2002). 2002.