Semantic Data Management: Towards Querying Data with their Meaning

Lipyeow Lim Haixun Wang Min Wang
IBM T. J. Watson Research Center
{liplim,haixun,min}@us.ibm.com

Abstract

Relational database management systems are constantly being extended and augmented to accommodate data in different domains. Recently, with the increasing use of ontology in various applications, the need to support ontology, especially the related inferencing operation, in DBMS has become more concrete and urgent. However, manipulating knowledge along with relational data in DBMSs is not a trivial undertaking due to the mismatch in data models. In this paper, we introduce a framework for managing relational data and hierarchical domain knowledge together. Our framework persists taxonomies contained in ontologies by leveraging XML support in hybrid relational-XML DBMSs (e.g., IBM's DB2 v9) and rewrites ontology-based semantic matching queries using the industry-standard query languages, SQL/XML and XQuery. Compared with previous approaches, our approach does not materialize transitive closures of ontological relationships to support inferencing. Consequently, our method has wide applicability and good performance.

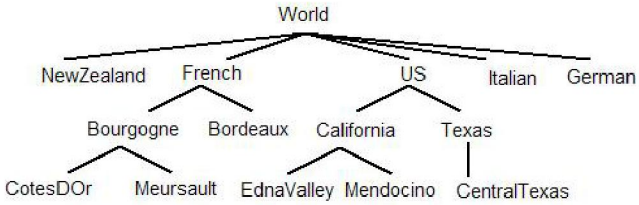
1 Introduction

Current DBMSs, albeit improved by many extensions over the years, are not ready to manipulate data closely with knowledge. On the other hand, an increasing number of applications are dealing directly with ontologies. An ontology is a machine readable vocabulary for specifying domain knowledge. Since the data are managed by DBMSs, it is desirable that the domain knowledge can be managed in the same framework, so that users can query the data, the domain knowledge, and the knowledge inferred from the data in the same way as querying just relational data. We call such an effort *semantic data management*.

In order to support semantic data management in DBMSs, new extensions are required to bridge the gap between data representation and knowledge representation/inferencing. Towards this goal, we propose a framework that leverages native XML capabilities in a DBMS to support inferencing operations such as subsumption checking. Before diving into the details of our method, we use an example to illustrate the task we are undertaking.

Id	Type	Origin	Maker	Price
1	Burgundy	CotesDOr	ClosDeVougeot	30
2	Riesling	NewZealand	Corbans	20
3	Zinfandel	EdnaValley	Elyse	15

(a) The Wine base table



(b) The locatedIn property.

Figure 1. Our running example consists of the wine base table, and the hierarchical locatedIn property from the wine ontology.

Running Example.. Assume we have a relational table for wines, as shown in Figure 1(a). Every row in the wine table is associated with a specific instance of wine. Each wine has the following attributes: type, origin, maker, and price.

Example 1 (Ontology-based Semantic Query) *To find wines that originated from US, we may naively issue the following SQL query:*

```
SELECT W.Id
FROM Wine AS W
WHERE W.Origin = 'US';
```

The above query fails to return the wines that semantically satisfy the query condition.

To provide semantically correct answers, the query processor must know (i) that "*Origin*" is a location, and (ii) what is location's semantics. In other words, we want to move from query-by-value to query-by-meaning.

Data semantics can be encoded in ontologies using several machine-readable formats. For the purpose of illustration, we

1-4244-0803-2/07/$20.00 ©2007 IEEE

will use the well-known wine ontology [13, 8] encoded in the Web Ontology Language (OWL) [10, 9]; however, we will limit our discussion to subsumptive relationships and defer the treatment of general, non-subsumptive relationships in our full paper. Consider the locatedIn property or relationship in the wine ontology as shown in Figure 1(b). In essence, the locatedIn relationship encodes a taxonomy of geographical regions along with the common-sense subsumption and transitive semantics.

Consider the query in Example 1 again. To use the locatedIn relationship from the ontology, instead of asking for wines whose origin *is equal to* US, we ask for wines whose origin is *located in* US. Suppose the query processor has access to the locatedIn relationship from the wine ontology as shown in Figure 1(b). Further suppose the query processor understands that the transitive semantics of the locatedIn relationship, that is,

$$locatedIn(a, b) \cap locatedIn(b, c) \Rightarrow locatedIn(a, c),$$

where a, b, c are object instances. The query processor can then determine that EdnaValley is located in California, and California is located in the US.

The preceding description shows the process of reasoning with ontology specific to Example 1. The goal of this paper is to provide a mechanism to assist the users to ask general, high level ontology-based queries against any dataset, and to enable the DBMSs to carry out inferencing over transitive relationships defined in the ontology such that these queries can be answered efficiently.

The first challenge in handling ontology-based semantics queries is how to store and access the ontology data, which are often represented by graphs. Another challenge is that relationships embodied by an ontology usually have a variety of properties, which include transitivity, equivalency, etc. A major focus in this paper is in handling transitive relationships, as they are involved in many useful queries (such as Example 1), but are difficult to express by the end user and are costly to process by the system, as they often require recursive SQL queries. Currently, to support ontology-based semantic queries efficiently in a DBMS, a well-known approach is to pre-process the ontology data by pre-computing and storing (materializing) the transitive closures for all transitive relationships in the ontology. The main problem with this approach is its huge time and storage overhead. Furthermore, it makes update of ontology data almost impossible once the transitive closures have been materialized.

Due to the challenges, we argue that neither relational databases nor pure XML databases are adequate for the task. In this paper, we show that ontology-based semantic queries can be naturally supported in a hybrid relational-XML DBMS (eg. IBM's DB2 Viper release).

Contributions. In summary, the contributions of this paper are as follows:

- We introduce a novel method for storing and manipulating ontology data in a hybrid relational-XML DBMS.
- We show that ontology-based subsumption queries can be expressed using industry-standard query languages, SQL/XML and XQuery, and they can be processed in a hybrid relational-XML DBMS using our storage model.
- Subsumption checking is one of the most expensive operation in processing queries over ontology data. Previous approaches pre-compute the transitive closure in order to achieve reasonable performance. Our method performs subsumption checking at query execution time by leveraging on XPath and XQuery support. Hence, our method does not suffer from the problem of storing large amounts of pre-computed inference results nor the problem of updating these pre-computed results when the ontology data change.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we briefly introduce hybrid database systems for relational and XML data using IBM DB2 v9 as an example. Section 3 describes how we store transitive relationships from ontology data, how semantic queries can be expressed, and how semantic queries can be re-written. In Section 4, we give a brief survey of previous work on supporting ontology-based semantic queries. Conclusions are drawn in Section 5.

2 Hybrid Relational-XML DBMSs

The logic modeling described in the previous section need physical level support. Our framework leverages hybrid relational-XML DBMSs to provide physical level support for domain knowledge. A hybrid relational-XML DBMS extends an RDBMS with the following components: (1) a native XML storage that stores an XML document as an instance of the XQuery Data Model (QDM), i.e., as a structured, typed, binary tree, (2) new index types for XML data including structural indexes, value indexes, and full-text indexes, (3) a hybrid query compiler that can process XQuery and SQL, and (4) an enhanced query runtime that supports XQuery and SQL/X operators.

In a hybrid relational-XML DBMS, XML is supported as a basic data type. Users can create a table with one or more XML type columns. A collection of XML documents can therefore be defined as a column in a table. For example, a user can create a table *transitiveReln* with the following statement:

```
CREATE TABLE transitiveReln(id integer,
                            name VARCHAR(27),
                            hierarchy XML);
```

Users can query relational columns and XML column together by issuing SQL/X query [3, 4]. For example, the following query returns relationship ids and names of all transitive relationships that contain the XPath `/World//CotesDOr`:

```
SELECT id, name
FROM transitiveReln AS T
WHERE XMLExists('$t/World//CotesDOr'
      PASSING BY REF T.hierarchy AS "t")
```

Note that `XMLExists` is an SQL/X boolean function that evaluates an XPath expression on an XML value. If XPath returns a nonempty sequence of nodes, then `XMLExists` is true, otherwise, it is false.

3 Our Framework

3.1 Storing Semantic Information

There are two obvious methods for storing ontology in DBMS. One method models ontologies as a collection of triples $\langle predicate, subject, object \rangle$, and store these triples in a RDBMS table. This approach is adopted by most ontology systems today [11, 2]. Alternatively, since ontologies are usually encoded in XML (eg. OWL format), we can store the ontology file as XML in an XML-enabled DBMS.

These two methods represent two extremes, and neither of them are effective in managing ontology data. If we use an RDBMS to manage all the triples, inferencing can only be accomplished by using recursive SQL queries or by precomputing the inferred triples. Take the geographical region hierarchy for example. Simple inquiries such as whether EdnaValley is located inside US can only be expressed by recursive queries. Recursive queries are hard to express and costly to process. Prematerialization of all inferred relationships also has problems. For the geographical hierarchy, it means we must store all pairs of locations as long as one is inside the other. This is often impractical for large ontologies and makes updates to the ontology extremely inefficient. The second method of storing the ontology file as XML naively has the disadvantage that it is quite difficult to write inferencing queries on the XML file.

The intuition behind our approach is to combine the strengths of XML database and RDBMS in handling different types of data. For instance, the subsumption relationship can be easily modeled by a tree structure (e.g., the "locatedIn" hierarchy in Figure 1(b)). It is well known that ancestor/decedent queries can be supported by tree labeling techniques in a very efficient way, which means we can avoid recursive queries or materialization if such relationships are stored in tree structures. On the other hand, ontology triples that normally do not participate in recursive inferencing can be handled by RDBMS very well, so we can use RDBMS for such data.

Our solution is to store ontology data in a hybrid relational-XML DBMS so that we can support ontology-based semantic queries efficiently.

For the transitive relationships addressed in this paper, we extract those relationships from ontology files and store them in the table:

```
transitiveReln(id INTEGER, name VARCHAR(27),
hierarchy XML)
```

Non-transitive relationships will be addressed in the full paper.

3.2 Expressing Semantic Queries

In order for the transitive relationships from the ontology that is stored in the `transitiveReln` table to be useful, we need to be able to combine the semantic information with existing relational data to answer semantic queries. There are several ways to express queries so that they exploit the semantic information in the transitive relationships and we outline them in this section.

3.2.1 Using SQL/XML

Leveraging on the native XML support, we can write SQL/XML queries that directly access the transitive relationships and the relational data. As an example, consider the locatedIn relationship of Figure 1(b). Assume we have stored the relationship in table `transitiveReln` following our discussion in Section 3.1.

Now, we can use the following SQL/XML query to find wines that originate from the United States (US).

Example 2 (Using SQL/XML directly) *Find all wines that originate from the US.*

```
SELECT W.Id
FROM wine AS W, transitiveReln AS T
WHERE XMLExists('$t//US//*
        [fn:string(node-name(.))=$r]'
        PASSING BY REF T.hierarchy AS "t",
        W.origin AS "r")
      AND T.id=2 AND T.name='locatedIn';
```

In the query, we use `XMLExists` to specify the location constraint in the ontology hierarchy that the query must satisfy. The query directly accesses the XML column in the `transitiveReln` table, and the user must use XPath expression in the query.

We study the pros and cons of expressing semantic queries in the form of Example 2. An ontology consists of a large variety of relationships. However, a majority of semantic queries are only concerned about a certain subsumption relationship, that is, whether two terms are related in the ontology hierarchy by relationships such as `locatedIn` or `is_a`. For instance, the above query is concerned about whether a region is a subregion of another region. Since this is the most frequent type of queries, we find that requiring users to write queries in the form of Example 2 has at least two disadvantages:

1. Queries are unnecessarily complicated even for simple tasks such as the one in Example 2. Users are forced to use XPath expressions inside SQL/XML even if their only purpose is about reachability between two nodes in an XML document.
2. Operators such as `XMLExists` are more on the procedural side. The lack of declarativeness will limit the room of

rule rewriting and optimization. This will eventually effect the performance. We address the importance of rule rewriting and the opportunity of optimization in more detail in our full paper.

3.2.2 Using the ONTOLOGY Operator

To overcome the above mentioned deficiency, we introduce a single ontology operator, ONTOLOGY, in the form of a table function. The operator provides not only syntactic sugar which aims at shielding users from the complexity in handling XML directly, but also gives the optimizer more room to rewrite it into more appropriate SQL/XML query forms. The table function takes three parameters:

```
ONTOLOGY(OID INT, R CHAR(20), T CHAR(20)),
```

where OID is the ontology ID, R is a relationship name, and T is a term. Conceptually, it returns a table that contains all terms X in the ontology identified by OID and R such that the relationship R(T, X) holds. For each term it returns, it also gives the distance between X and T in the corresponding ontology hierarchy (i.e., the corresponding XML document) [1].

However, it does not necessarily mean that an invocation of the ONTOLOGY table function will materialize the table as the one shown above. Instead, the rule rewriting and the optimization mechanism will decide how to map the query in the most efficient way into XPath expressions that access the ontology. We address this issue in more detail in our full paper.

Now, with the ONTOLOGY table function, we can express the query in Example 2 in a more succinct and declarative manner.

Example 3 (Using the ONTOLOGY operator) *Find all wines that originate from US.*

```
SELECT W.id
FROM wine AS W,
     TABLE (ONTOLOGY(2,'locatedIn','US')) AS
     T
WHERE T.term = W.origin;
```

The table function ONTOLOGY can also take an optional parameter, *reverse*. When *reverse* is *True*, the invocation ONTOLOGY(OID, R, T, *True*) will return all terms X such that the relationship R(X, T) holds. For instance, calling ONTOLOGY(2, subClassOf, 'EdnaValley', *True*) will return California and US as matching terms. This allows us to traverse the subClassOf hierarchy in the reverse direction when inferencing.

3.2.3 Using views

If the association between a transitive relationship and the relational data is more long term (beyond a single ad-hoc query), a view can also be created that associates the transitive relationship with the relational data.

Example 4 (Using views) *To find wines that originated from US, we first create a view to contain the transitive closure of the origin,*

```
CREATE VIEW OriginView(Id, OriginLocatedIn) AS
SELECT W.Id, T.term
FROM Wine AS W,
     TABLE (ONTOLOGY(2, 'locatedIn',
     W.Origin, true)) AS T
```

and issue the following SQL query on the view:

```
SELECT V.Id
FROM OriginView AS V
WHERE V.OriginLocatedIn = 'US'
```

The advantages of using a view is that because the association is longer term and not per query, more optimization opportunities are available.

3.3 Re-writing Semantic Queries

Section 3.2 has already described the SQL/XML XMLExists function and showed how it can be used as one implementation of the ONTOLOGY table function. In addition to the XMLExists function, we describe another SQL/XML function, XMLTable, that can be used to implement the ONTOLOGY table function as well. The XMLTable function creates a virtual relational table using information from XML data specified using XPath.

We illustrate using an examples how queries written using the ONTOLOGY table function can be re-written in SQL/XML using the XMLExists and XMLTable functions. Without knowledge of the implementation of these SQL/XML functions in the query execution engine, it is not possible to decide which re-writing will be more efficient; however, we will discuss optimization issues in our full paper.

Example 5 (Query Rewrite) *Consider our earlier example to find all wines that originate from US:*

```
SELECT W.id
FROM wine AS W,
     TABLE (ONTOLOGY(2,'locatedIn', 'US'))
     AS T
WHERE T.term = W.origin;
```

There are at least two ways to express this query in the SQL/XML language. The query can be expressed using the XMLExist boolean operator:

```
SELECT W.id
FROM wine AS W, transitiveReln AS T
WHERE XMLExists('$t//US
        //*[fn:string(node-name(.))=$r]'
        PASSING BY REF T.hierarchy
        AS "t", W.origin AS "r")
      AND T.id=2 AND T.name='locatedIn';
```

[1] The distance column is an optional column that may be used for query result ranking in some applications. The distance values can be computed by annotating each XML node by its level in the pre-processing step.

Conceptually, this rewritten query iterates through each row of the `wine` table and checks if the origin field is a descendant of US in the locatedIn XML tree of the `transitiveReln` ontology table. Alternatively, the query can also be re-written using the `XMLTable` table function:

```
SELECT W.id
FROM wine AS W, transitiveReln AS T,
     XMLTable('$t//US//*'
       PASSING T.hierarchy AS "t"
       COLUMNS "ename" VARCHAR(80)
       PATH 'fn:string(node-name(.))'  AS X
WHERE T.id=2 AND T.name='locatedIn'
      AND X."ename" = W.origin;
```

Conceptually, this query creates a virtual table X with a single column `ename` that contains all the subregions of US and performs a join on the origin column of the `wine` table and the virtual table.

For the case of views, the queries can be re-written in a similar way.

4 Related Work

Several systems have been developed for building and manipulating ontologies. For example, Protégé is an ontology editor and a knowledge-base editor that allows the user to construct a domain ontology, customize data entry forms, and enter data [12]. RStar is an RDF storage and query system for enterprise resource management [5]. Other ontology building systems include OntoEdit [7], OntoBroker [6], OntologyBuilder and OntologyServer [1], and KAON [11]. Most systems use a file system to store ontology data (e.g., OntoEdit). Others (e.g., RStar and KAON) allow the ontology data to be stored in a relational DBMS. However, processing of ontology-related queries in these systems is typically done by an external middle-ware (wrapper) layer built on top of a DBMS engine. Two key limitations of this loosely-coupled approach are: (1) DBMS users cannot reference ontology data directly, and (2) query processing of ontology-related queries cannot leverage the the query processing and optimization power of a DBMS.

The most recent advance in ontology management in the database community is a novel approach by Oracle [2]. In [2], Das et al. propose a method to support ontology-based semantic matching in RDBMS using SQL directly. Ontology data are pre-processed and stored in a set of system-defined tables. A set of special operators and a new indexing scheme are introduced. A database user can thus reference the ontology data directly using the new operators. Compared to the loosely-coupled approach, this method opens up the possibility of combining ontology query operators with existing SQL operators such as joins. The ability to manipulate ontology data and regular relational data directly in the RDBMS greatly simplifies and facilitates the development of ontology-driven applications.

However, due to the "mismatch" between the relational schema and the graphical model of ontology data, this relational-model based approach is still quite limited in its query processing efficiency. For example, inference is the most expensive operation on ontology data when supporting semantic matching queries. All the previous approaches (including the approach in [2]) need to pre-compute and materialize all (or a big part of) the inference results (i.e., transitive closures) to achieve reasonable performance at query execution time. This pre-processing not only incurs serious time and storage overhead, but also makes the update of the pre-computed data infeasible when the underlying ontology data changes.

5 Conclusion

In this paper we have described methods to store, manage ontology data using a hybrid relational-XML DBMS without the need to materialize transitive closure for inferencing. We have showed how ontology data can be queried together with relational data using the SQL/XML query language in conjunction with the XPath specification. Our work showed that DBMSs with native XML support is a good candidate to support ontology-based queries.

References

[1] A. Das, W. Wu, and D. L. McGuinness. Industrial strength ontology management. In *The Emerging Semantic Web*, 2001.

[2] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting ontology-based semantic matching in RDBMS. In *VLDB*, pages 1054–1065, 2004.

[3] A. Eisenberg and J. Melton. SQL/XML is making good progress. *SIGMOD Record*, 31(2):101–108, 2002.

[4] A. Eisenberg and J. Melton. Advancements in SQL/XML. *SIGMOD Record*, 33(3):79–86, 2004.

[5] L. Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu. RStar: An RDF storage and query system for enterprise resource management. In *CIKM*, 2004.

[6] OntoBroker. `http://ontobroker.aifb.uni-karlsruhe.de/index_ob.html`.

[7] OTK tool repository: Ontoedit. `http://www.ontoknowledge.org/tools/ontoedit.shtml`.

[8] OWL web ontology language guide, February 2004. `http://www.w3.org/TR/owl-guide/`.

[9] OWL web ontology language overview, February 2004. `http://www.w3.org/TR/owl-features/`.

[10] OWL web ontology language reference, February 2004. `http://www.w3.org/TR/owl-ref/`.

[11] The KArlsruhe ONtology and semantic web tool suite. `http://kaon.semanticweb.org/`.

[12] The protg ontology editor and knowledge acquisition system. `http://protege.stanford.edu/`.

[13] Wine ontology. `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`.