Supporting Ontology-based Semantic Matching in RDBMS

Souripriya Das, Eugene Inseok Chong, George Eadon, Jagannathan Srinivasan
Oracle Corporation
One Oracle Drive, Nashua, NH 03062, USA

Abstract

Ontologies are increasingly being used to build applications that utilize domain-specific knowledge. This paper addresses the problem of supporting ontology-based semantic matching in RDBMS. Specifically, 1) A set of SQL operators, namely ONT_RELATED, ONT_EXPAND, ONT_DISTANCE, and ONT_PATH, are introduced to perform ontology-based semantic matching, 2) A new indexing scheme ONT_INDEXTYPE is introduced to speed up ontology-based semantic matching operations, and 3) System-defined tables are provided for storing ontologies specified in OWL. Our approach enables users to reference ontology data directly from SQL using the semantic match operators, thereby opening up possibilities of combining with other operations such as joins as well as making the ontology-driven applications easy to develop and efficient. In contrast, other approaches use RDBMS only for storage of ontologies and querying of ontology data is typically done via APIs. This paper presents the ontology-related functionality including inferencing, discusses how it is implemented on top of Oracle RDBMS, and illustrates the usage with several database applications.

1. Introduction

An ontology is a shared conceptualization of knowledge in a particular domain. It facilitates building applications by separating knowledge about the target domain from the rest of the application code. The key benefits of this approach are: simplification of the application code, possible sharing of knowledge among multiple applications, and the flexibility of evolving the knowledge without requiring changes to the application.

This approach has been used to build applications for various domains (such as clinical applications [3], geographic information system [2], integrated knowledge management [1], and knowledge acquisition system [7]). The same approach can be adopted to build database applications. This paper addresses the problem of supporting ontology-based semantic matching in RDBMS.

To motivate the need for ontology-based semantic matching, consider a restaurant guide application, which recommends restaurants to a user based on her/his preferences. Consider a table `served_food` that contains the types of cuisines served at restaurants.

Table 1: served_food

R_id	Cuisine
1	American
2	Mexican
2	American
14	Portuguese

In the absence of semantic matching, the application would most likely resort to syntactic matching via the `‘=’` operator as shown below:

```
SELECT * FROM served_food
WHERE cuisine = ‘Latin American’;
```

This query generates no rows since none of Cuisine values in the table will match `‘Latin American’`.

In contrast, the user can get more meaningful results by performing semantic matching that consults an ontology (such as the cuisine ontology in Figure 1) for computing the results. Specifically, a user can issue the following query:

```
SELECT * FROM served_food
WHERE ONT_RELATED(cuisine,
                  ‘IS_A’,
                  ‘Latin American’,
                  ‘Cuisine_ontology’)=1;
```

Here the `ONT_RELATED` operator determines if the two input terms are related by the input relationship type argument by consulting the specified ontology. If they are related, then the operator will return 1, otherwise 0.

Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004

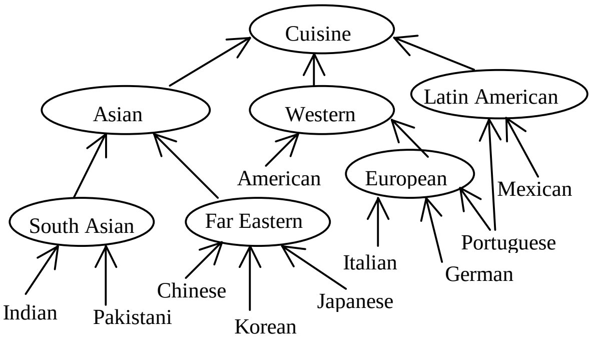


Figure 1: A Cuisine Ontology
(Each node represents an Individual and each edge represents `a transitive` *ObjectProperty* `'IS_A')`

The query identifies rows containing cuisines that are related to `'Latin American'` based on `'IS_A'` relationship. The query will generate restaurants 2 and 14 since `'Mexican'` and '`Portuguese`' are related to `'Latin American'` cuisine. Thus, one can incorporate semantics of the particular knowledge domain in SQL queries by introducing ontology-based semantic matching.

Optionally, a user may want to get a measure for the rows filtered by `ONT_RELATED` operator. This can be achieved by using `ONT_DISTANCE` ancillary operator. The `ONT_DISTANCE` operator gives a measure of how closely the terms are related by measuring the distance between the two terms. Continuing with the example, one can get the result sorted on distance measure as follows:

```
SELECT * FROM served_food
WHERE ONT_RELATED (cuisine,
                   'IS_A',
                   'Latin American',
                   'Cuisine_ontology',
                    123) = 1
ORDER BY ONT_DISTANCE (123[1]);
```

Similarly, another ancillary operator `ONT_PATH` would be useful, which computes path information between the two terms.

In addition, a user may want to query an ontology independently (without involving user tables). The `ONT_EXPAND` operator can be used for this purpose (see Section 2 for details).

Providing ontology-based semantic matching capability as part of SQL greatly facilitates developing ontology-driven database applications. Applications that can benefit include e-commerce (such as supply chain management, application integration, personalization, and auction). Also, applications that have to work with domain-specific knowledge repositories (such as BioInformatics, Geographical Information Systems, and Healthcare Applications) can take advantage of this capability. These capabilities can be exploited to support semantic web applications (such as web service discovery [8]) as well. A key requirement in these applications is to provide semantic matching between syntactically different terms or sometimes between syntactically same, but semantically different terms [10].

Another category of the semantic matching application is related to knowledge reuse. Neutral Authoring [9] is an application area, where information is represented in a single language and then converted into different languages for multiple target systems. Corporate knowledge bases on which documentation and software development can be based is another big application area of such a capability.

Support for ontology-based semantic matching is achieved by introducing the following:

- Two new SQL operators, `ONT_RELATED` and `ONT_EXPAND` (as described above) are defined to model ontology-based semantic matching operations. For queries involving `ONT_RELATED` operator, two ancillary SQL operators `ONT_DISTANCE` and `ONT_PATH`, are defined that return distance and path respectively for the filtered rows. Additional operators may be introduced for querying purposes, e.g., for finding datatype property values.
- A new indexing scheme `ONT_INDEXTYPE` is defined to speed up ontology-based semantic matching operations.
- A schema has been designed to store information extracted from an ontology. This schema is not directly visible to the user.

The proposed functionality can be implemented by exploiting the database extensibility capabilities (namely, the ability to define user-defined operators, user-defined indexing schemes, and table functions) typically available in an RDBMS.

We support ontologies specified in Web Ontology Language (OWL [19], specifically, OWL Lite and OWL DL) by extracting information from the OWL document and then storing this information in the schema.

Oracle's Extensibility Framework [5] has been used to implement the operators and the new indexing scheme. Specifically, ONT_RELATED, ONT_DISTANCE, and ONT_PATH operators are implemented as user-defined operators. ONT_EXPAND is implemented as a table function. The operator implementation typically requires computing a transitive closure based upon explicit relationships and inferred relationships. This is performed via queries with CONNECT BY clause. The ONT_INDEXTYPE is implemented as a user-defined indexing scheme (See Section 3.3 for details).

[1] This argument identifies the filtering operator expression (ONT_RELATED) that computes this ancillary value [11].

1.1 Related Work

Ontologies have been around for sometime and in recent years they have received wider attention in the context of semantic web [4]. Several ontology building tools have been developed (for example, OntoEdit [13], OntoBroker [14], OntologyBuilder and OntologyServer [16], KAON [15]). Most of these tools use a file system to store ontologies. Among these, KAON and VerticalNet products (OntologyBuilder and OntologyServer) as well as the Jena2 Semantic Web Framework [21] allow storing of ontology using RDBMS and they provide an API for access and manipulation of ontologies. However, the key difference is that our approach makes ontology-based semantic matching available as part of SQL.

1.2 Organization of the Paper

Section 2 presents a feature overview of supporting ontology-based semantic matching operations. Section 3 discusses the implementation of the ontology-related changes on top of Oracle RDBMS and its performance results. Section 4 illustrates their usage with several database applications. Section 5 describes our experience and Section 6 concludes with a summary and outlines future work.

2. Supporting Ontology-based Semantic Matching on Oracle RDBMS

Although the feature is described in the context of Oracle RDBMS, it can be supported in any RDBMS that support a few basic extensibility capabilities as outlined in Section 1.

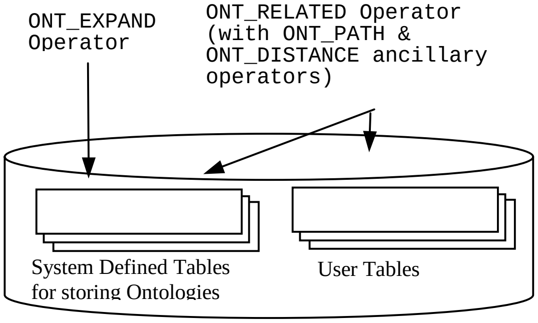

ONT_EXPAND Operator
ONT_RELATED Operator (with ONT_PATH & ONT_DISTANCE ancillary operators)
System Defined Tables for storing Ontologies
User Tables

Figure 2: Ontology-related Functionality

2.1 Overview

The ontology-related functionality (see Figure 2) is as follows:

- An RDBMS schema, consisting of several system-defined tables (see Section 2.2 for details), is created for storing information extracted from the ontologies.
- Two operators are provided for querying purposes. The `ONT_EXPAND` operator can be used to query the ontology independently, whereas `ONT_RELATED` operator can be used to perform queries on a user table holding ontology terms.
- Optionally, a user can use two (ancillary) operators, `ONT_DISTANCE` and `ONT_PATH`, in queries involving the `ONT_RELATED` operator to get additional measures (distance and path) for the filtered rows.

These are further elaborated below.

2.2 RDBMS Schema for Storing Ontologies

An RDBMS schema has been created for storing ontologies specified in OWL. The tables in this schema include:

```
Ontologies (
  OntologyID, OntologyName, Owner, ...)
```

```
Terms (
  TermID, OntologyID, Term, Type, ...)

Properties (
  OntologyId, PropertyID,
  DomainClassID, RangeClassID,
  Characteristics, ...)

Restrictions (
  OntologyID, NewClassID, PropertyID,
  MinCardinality, MaxCardinality,
  SomeValuesFrom, AllValuesFrom, ...)
```

```
Relationships (
  OntologyID, TermID1, PropertyID,
  TermID2, ...)

PropertyValues (
  OntologyID, TermID, PropertyID,
  Value, ...)
```

- ***Ontologies***: Contains basic information about various ontologies.
- ***Terms***: Represents classes, individuals, and properties in the ontologies. A term is a lexical representation of a concept within an ontology. `TermID` value is generated to be unique across all ontologies. This allows representation of references to a term in a different ontology than the one that defines the term. Also, even an `OntologyID` is handled as a `TermID` which facilitates storing values for various properties (e.g., Annotation Properties) and other information that applies to an ontology itself. Note that, as a convention, any column in the above schema whose name is of the form "…ID..", would actually contain `TermID` values (like a foreign key).
- ***Properties***: Contains information about the properties. Domain and range of a property are represented with `TermID` values of the corresponding classes. `Characteristics` indicate

which of the following properties are true for the property: symmetry, transitivity, functional, inverse functional.

- ***Restrictions***: Contains information about property restrictions. Restrictions on a property results in definition of a new class. This new class is not necessarily named (i.e., 'anonymous' class) in OWL. However, internally we create a new (system-defined) class for ease of representation.
- ***Relationships***: Contains information about the relationship between two terms.
- ***PropertyValues***: Contains <Property, Value> information associated with the terms. In order to handle values of different data types, some combinations of the following may be used: Define separate tables (or separate columns in the same table) for each of the frequently encountered types and use a generic self-describing type (ANYDATA in Oracle RDBMS) to handle any remaining types.

System-defined Classes for Anonymous Classes: We create internal (i.e., not visible to the user) or system-defined classes to handle OWL anonymous classes that arise in various situations such as Property Restrictions, enumerated types (used in DataRange), class definitions expressed as expression involving IntersectionOf, UnionOf, and ComplementOf.

Bootstrap Ontology: The first things that are loaded into the above schema are the basic concepts of OWL itself. In some sense this is like the bootstrap ontology. For example:

- Thing and Nothing are stored as Classes.
- subClassOf is stored as a transitive (meta) property that relates two classes.
- subPropertyOf is stored as a transitive (meta) property that relates two properties.
- disjointWith is stored as a symmetric (meta) property that relates two classes.
- SameAs is stored as a transitive and symmetric property that relates two individuals in Thing class.

Storing these OWL concepts as a bootstrap ontology facilitates inferencing. A simple example would be the following: If C1 is a subclassOf C2 and C2 is a subclassOf C3, then (by transitivity of subClassOf) C1 is a subclassOf C3. Note that the reflexive nature of subClassOf and SubPropertyOf is handled as a special case.

Loading Ontologies: An ontology is loaded into the database by using an API that takes as input an OWL document. Information from the OWL document is extracted and then stored into the system-defined tables in the RDBMS schema described above.

The `Ontologies` table stores some basic information about all the ontologies that are currently stored in the database. A portion (view) of this table is visible to the user.

2.3 Modeling Ontology-based Semantic Matching

To support ontology-based semantic matching in RDBMS several new operators are defined.

2.3.1 ONT_RELATED Operator. This operator models the basic semantic matching operation. It determines if the two input terms are related with respect to the specified `RelType` relationship argument within an ontology. If they are related it returns 1, otherwise it returns 0.

```
ONT_RELATED (Term1, RelType, Term2,
             OntologyName
) RETURNS INTEGER;
```

The `RelType` can specify a single ObjectProperty (for example, `'IS_A'`, `'EQV'`, etc.) or it can specify a combination of such properties by using `AND`, `NOT`, and `OR` operators (for example, `'IS_A OR EQV')`. Note that both `Term1` and `Term2` need to be simple terms. If `Term2` needs to be complex involving AND, OR, and NOT operators, user can issue query with individual terms and combine them with INTERSECT, UNION, and MINUS operators. See Section 2.3.4 for an example.

`RelType` specified as an expression involving `OR` and `NOT` operators (e.g., `FatherOf OR MotherOf`) is treated as a virtual relationship (in this case say `AncestorOf`) that is transitive by nature (also see Section 3.2.5).

2.3.2 ONT_EXPAND Operator. This operator is introduced to query an ontology independently. Similar to `ONT_RELATED` operator, the `RelType` can specify either a simple relationship or combination of them.

```
CREATE TYPE ONT_TermRelType AS OBJECT (

 Term1Name VARCHAR(32),
 PropertyName VARCHAR(32),
  Term2Name VARCHAR(32),

  TermDistance NUMBER,
  TermPath VARCHAR(2000)
);

CREATE TYPE ONT_TermRelTableType AS
  TABLE OF ONT_TermRelType;

ONT_EXPAND (Term1, RelType, Term2,
           OntologyName
) RETURNS ONT_TermRelTableType;
```

Typically, non-NULL values for `RelType` and Term2 are specified as input and then the operator computes all the appropriate <Term1, RelType, Term2> tuples in the closure taking into account the characteristics (transitivity and symmetry) of the specified `RelType`. In addition, it also computes the relationship measures in terms of distance (TermDistance) and path (TermPath). For cases when a term is related to input term by multiple paths, one row per path is returned. It is also possible that ONT_EXPAND invocation may specify input values for any one or more of the three parameters or even none of the three parameters. In each of these cases, the appropriate set of <Term1, RelType, Term2> tuples is returned.

2.3.3 ONT_DISTANCE and ONT_PATH Ancillary Operators. These operators compute the distance and path measures respectively for the rows filtered using ONT_RELATED operator.

```
ONT_DISTANCE (NUMBER) RETURNS NUMBER;
ONT_PATH (NUMBER) RETURNS VARCHAR;
```

A single resulting row can be related in more than one way with the input term. For such cases, the above operators return the optimal measure, namely smallest distance or shortest path. For computing all the matches, the following two operators are provided:

```
ONT_DISTANCE_ALL (NUMBER)
 RETURNS TABLE OF NUMBER;
ONT_PATH_ALL (NUMBER)
 RETURNS TABLE OF VARCHAR;
```

2.3.4 A Restaurant Guide Example. Consider a restaurant guide application that maintains type of cuisine served at various restaurants. It has two tables, 1) restaurants containing restaurant information, and 2) served_food containing the types of cuisine served at restaurants.

The restaurant guide application takes as input a type of cuisine and returns the list of restaurants serving that cuisine. Obviously, applications would like to take advantage of an available cuisine ontology to provide better match for the user queries. The cuisine ontology describes the relationships between various types of cuisines as shown earlier in Figure 1.

Thus, if a user is interested in restaurants that serve cuisine of type 'Latin American', the database application can generate the following query:

```
SELECT r.name, r.address
FROM served_food sf, restaurant r
 WHERE r.id = sf.r_id AND
       ONT_RELATED(sf.cuisine,
                   'IS_A OR EQV',
                  'Latin American',
                   'Cuisine_ontology')=1;
```

To query on 'Latin American' AND 'Western' the application program can obtain rows for each and use the SQL INTERSECT operation to compute the result.

Also, the application can exploit the full SQL expressive power when using ONT_RELATED operator. For example, it can easily combine the above query results with those restaurants that have lower price range.

```
SELECT r.name
FROM served_food sf, restaurant r
 WHERE r.id = sf.r_id AND
       ONT_RELATED(sf.cuisine,
                    'IS_A OR EQV',
                   'Latin American',
                    'Cuisine_ontology')=1
        AND r.price_range = '$';
```

2.3.5 Discussion. Note that the queries in section 2.3.4 can also be issued using the ONT_EXPAND operator. For example, the first query in that section can alternatively be expressed using ONT_EXPAND as follows:

```
SELECT r.name, r.address
FROM served_food sf, restaurant r
 WHERE r.id = sf.r_id AND
       sf.cuisine IN
       (SELECT Term1Name from TABLE(
        ONT_EXPAND(NULL, 'IS_A OR EQV',
                   'Latin American',
                  'Cuisine_ontology')));
```

The ONT_RELATED operator is provided in addition to ONT_EXPAND operator for the following reasons:

- The ONT_RELATED operator provides a more natural way of expressing semantic matching operations on column holding ontology terms.
- It allows use of an index created on column holding ontology terms to speed up the query execution by taking column data into account.

2.4 Inferencing

Inferencing rules employing the symmetry and transitivity characteristics of properties are used to infer new relationships. This kind of inferencing can be achieved through the use of the operators defined above (see Section 3.2 for details). Note that our support for inferencing is restricted to OWL Lite and OWL DL, both of which are decidable.

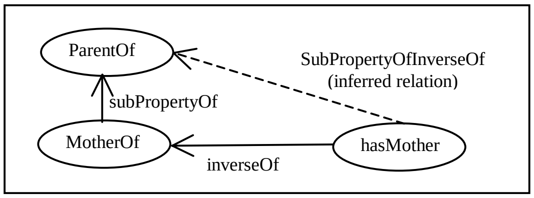
To support more complete inferencing using the above operators, an initial phase of inferencing is done after ontology is loaded, and the results of this inferencing are stored persistently and used in subsequent inferencing. Several examples of the initial inferencing follow.

All subPropertyOf relationships are derived and stored during this phase. The transitive aspects of sameAs (e.g., sameAs(x,y) AND sameAs(y,z) IMPLIES sameAs(x,z)) can be handled by the operators defined above. However, the more complex rules which imply sameAs (e.g., p is a functional property AND p(a,x) AND p(b,y) AND sameAs(a,b) IMPLIES sameAs(x,y)) are best handled outside of the operator implementation. We expect these complex sameAs inferences to be done during the initial inferencing phase. To provide the semantics of sameAs during closure computation, the operators will treat an individual 'I' as 'I OR J' for all J where sameAs(I, J).

Furthermore, we are introducing an internal relationship that will be useful for inferencing over complex subPropertyOf and inverseOf interactions: SubPropertyOfInverseOf(f,g) iff f(x,y) IMPLIES g(y,x). Again, we expect that the rules that introduce

SubPropertyOfInverseOf (spiOf, for short) into an ontology (e.g., inverseOf(f,g) IMPLIES spiOf(f,g) AND spiOf(g,f)) will be handled during the initial inferencing phase. However, the transitive aspects of spiOf can be handled as a special case within our operators, according to the following rules:

- subPropertyOf(f,g) AND spiOf(g,h)
  IMPLIES spiOf(f,h)
  (Proof: f(x,y) IMPLIES g(x,y) IMPLIES h(y,x))
- spiOf(f,g) AND subPropertyOf(g,h)
  IMPLIES spiOf(f,h)
  (Proof: f(x,y) IMPLIES g(y,x) IMPLIES h(y,x))
- spiOf(f,g) AND spiOf(g,h)
  IMPLIES SubPropertyOf(f,h)
  (Proof: f(x,y) IMPLIES g(y,x) IMPLIES h(x,y))



Given the expansion of subPropertyOf and spiOf, we can find all relationship tuples for a given property, including those that are implied through sub-properties and inverse-properties. Consider the following example (see adjoining figure), where non-transitive properties are used for clarity, which expands ParentOf. In this case, if we have subPropertyOf(MotherOf, ParentOf) and inverseOf(MotherOf, hasMother), we will get spiOf(hasMother, ParentOf) based on result of the initial inferencing phase and the above rules. Then hasMother(child, mother) will be sufficient to yield ParentOf(mother, child) in the result set:

```
SELECT r.termID1, 'ParentOf', r.termID2
FROM relationships r, terms t
WHERE r.propertyID = t.termID
  AND t.term IN
 (select term1Name FROM
  TABLE(ONT_EXPAND(NULL,
                   'subPropertyOf',
                   'ParentOf',
                   'Family_ontology')))
UNION
SELECT r.termID2, 'ParentOf', r.termID1
FROM relationships r, terms t
WHERE r.propertyID = t.termID
  AND t.term IN
 (select term1Name FROM
  TABLE(ONT_EXPAND(NULL,
                   'spiOf',
                   'ParentOf',
                   'Family_ontology')))
```

3. Implementation of Ontology Related Functionality on Oracle RDBMS

This section describes how the ontology-related functionality is implemented on top of Oracle RDBMS.

3.1 Operators

The `ONT_RELATED` operator is defined as a *primary* user-defined operator, with `ONT_DISTANCE` and `ONT_PATH` as its *ancillary* operators. The primary operator computes the ancillary value as part of its processing [11]. In this case, `ONT_RELATED` operator computes the relationship. If ancillary values (the distance and path measure) are required, it computes them as well.

Note that the user-defined operator mechanism in Oracle allows for sharing state across multiple invocations. Thus, the implementation of the `ONT_RELATED` operator involves building and compiling an SQL query with CONNECT BY clause (as described in Section 3.2) during its first invocation. Each subsequent invocations of the operator simply uses the previously compiled SQL cursor, binds it with the new input value, and executes it to obtain the result.

The `ONT_EXPAND` operator is defined as a table function as it returns a table of rows, which by default includes the path and distance measures.

3.2 Basic Algorithm

Basic processing for both `ONT_RELATED` and `ONT_EXPAND` involves computing transitive closure, namely, traversal of a tree structure by following relationship links given a starting node. Also, as part of transitive closure computation, we need to track the distance and path information for each pair formed by starting node and target node reached via the relationship links.

Oracle supports transitive closure queries with CONNECT BY clause as follows:

```
SELECT … FROM … START WITH <condition>
CONNECT BY <condition>;
```

The starting node is selected based on the condition given in START WITH clause, and then nodes are traversed based on the condition given in CONNECT BY clause. The parent node is referred to by the PRIOR operator. For computation of distance and path, the Oracle-provided LEVEL psuedo-column and SYS_CONNECT_BY_PATH function are respectively used in the select list of a query with CONNECT BY clause.

Note that in the system-defined *Relationships* table, a row represents ‘TermID1 is related to TermID2 via PropertyID relationship.’ For example, if ‘A IS_A B’, it is represented as the row <1, A, IS_A , B> assuming that the ontologyID is 1.

Note that any cycles encountered during the closure computation will be handled by the CONNECT BY NOCYCLE query implementation available in Oracle 10g (not explicitly shown in the examples below). Also, the proposed index-based implementation (described in

Section 3.3) can handle this case even in Oracle 9i Release 2.

For simplicity, we use a slightly different definition for the `relationships` table where term names are stored instead of termIDs as follows:

```
Relationships (
  OntologyName, Term1, Relation, Term2,
  ...)
```

To illustrate the processing, we use the restaurant guide example. The data for the two tables used are as shown in Table 2 below.

Table 2: Example Restaurant Database

restaurant

Id	Name	price_range	……
1	Mac	$	
2	Chilis	$$	
3	Anthonys	$$$	
4	BK	$	
5	Uno	$$	
6	Wendys	$	
7	Dabin	$$	
8	Cheers	$$	
9	KFC	$	
10	Sizzlers	$$	
11	Rio	$$	
12	Maharaj	$$	
13	Dragon	$$	
14	Niva	$$	

served_food

R_id	cuisine
1	American
2	Mexican
2	American
3	American
4	American
5	American
5	Italian
6	American
7	Korean
7	Japanese
8	American
9	American
10	American
11	Brazilian
12	Mexican
12	Indian
13	Chinese
14	Portuguese

3.2.1 Handling Simple Terms. Consider a query that has simple relation types, i.e., no AND, OR, NOT operators. The first query given in Section 2.3.4 can be converted as follows:

Original Query:

```
   SELECT r.name, r.address
   FROM served_food sf, restaurant r
   WHERE r.id = sf.r_id AND
         ONT_RELATED(sf.cuisine, ‘IS_A’,
 ‘Latin American’,
 ‘Cuisine_ontology’)=1;
```

Transformed Query:

```
  SELECT r.name, r.address
  FROM served_food sf,restaurant r
  WHERE r.id = sf.r_id AND
       sf.cuisine IN
      (SELECT term1 FROM relationships
       START WITH
        term2 = 'Latin American' AND
        relation = ‘IS_A’
      CONNECT BY
        PRIOR term1 = term2 AND
        relation = 'IS_A');
```

The text in boldface above is the portion that has been converted. Basically, the third argument is translated into START WITH clause and the second argument into CONNECT BY clause.

The result for this query is as follows:

```
NAME                ADDRESS
Chilis              ......
Maharaj             ......
Niva                ......
```

3.2.2 Handling OR Operator. Consider a case where `‘Brazilian’` cuisine was not originally included in the ontology and is now inserted under the `‘South American’` cuisine. Also, to put `‘South American’` cuisine in the same category as `‘Latin American’` cuisine, the transitive and symmetric `‘EQV’` relationship is used as shown in the Figure 3 below:

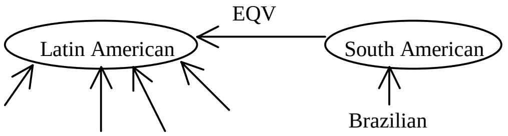


Figure 3: Adding EQV Relationship

Now, to get `‘Latin American’` cuisine, disjunctive conditions should be used to traverse both relationship links, that is, `‘IS_A’` and `‘EQV’`. Such disjunctive conditions can be directly specified in the START WITH and CONNECT BY clauses.

Original Query:

```
  SELECT r.name, r.address
   FROM served_food sf, restaurant r
    WHERE r.id = sf.r_id AND
          ONT_RELATED(sf.cuisine,
                      ‘IS_A OR EQV’,
                      ‘Latin American’,
                      ‘Cuisine_ontology’)=1;
```

Transformed Query:

The only differences from the transformed query of the previous example is that the relationships table:

```
FROM relationships
```

is replaced by a sub-query to introduce the implicit symmetric edges into the query:

```
FROM (SELECT term1, relation, term2
      FROM relationships
      UNION
      SELECT term2, relation, term1
      FROM relationships
      WHERE relation = ‘EQV’)
```

and the occurrence of the following predicate in START WITH and CONNECT BY clauses

```
relation = ‘IS_A’
```

is replaced with the following predicate:

```
    (relation = ‘IS_A’ OR relation =
    ‘EQV’)
```

3.2.3 Handling AND operator. Conjunctive conditions between transitive relationship types can be handled by independently computing the transitive closure for each relationship type and then applying set INTERSECT on the resulting sets. For each node in the intersection, a path exists from the start node to this node for each relationship type and hence this is sufficient.

Let us consider another relationship between cuisines, which identifies the spiciest cuisine using the term MOST_SPICY. The ontology can now contain information such as ‘South Asian cuisine is MOST_SPICY Asian cuisine’ and ‘Indian cuisine is MOST_SPICY South Asian cuisine,’ etc.

To find very spicy cuisine from the ontology, user can issue a query using conjunctive conditions in the relationships as follows:

Original Query: Find a restaurant that serves very spicy
Asian cuisine.

```
  SELECT r.name FROM served_food sf,
                     restaurant r
   WHERE r.id = sf.r_id AND
   ONT_RELATED(sf.cuisine,
              ‘IS_A AND MOST_SPICY’
              ‘Asian’,
               ‘Cuisine_ontology’) = 1;
```

Transformed query:

```
  SELECT r.name FROM served_food sf,
                restaurant r
    WHERE r.id = sf.r_id AND
        sf.cuisine IN
     (
      SELECT term1 FROM relationships
       START WITH
         term2 = 'Asian' AND
         relation = ‘IS_A’
      CONNECT BY
         PRIOR term1 = term2 AND
         relation = 'IS_A'

     INTERSECT

      SELECT term1 FROM relationships
       START WITH
         term2 = 'Asian' AND
         relation = ‘MOST_SPICY’
       CONNECT BY
          PRIOR term1 = term2 AND
           relation =‘MOST_SPICY’);
```

3.2.4 Handling NOT operator. A NOT operator specifies which relationships to exclude when finding transitive closure. Therefore, given the start node all relationships except ones specified in NOT operator will be traversed. NOT operators can be directly specified in the START WITH and CONNECT BY clauses.

Original Query: Find all Latin American cuisine, excluding ‘EQV’ relationship types.

```
  SELECT r.name FROM served_food sf,
                     restaurant r
   WHERE r.id = sf.r_id AND
         ONT_RELATED(sf.cuisine,
                     ‘NOT EQV’,
                     ‘Latin American’,
                     ‘Cuisine_ontology’)=1;
```

Transformed Query: Only difference from the transformed query of the example in Section 3.2.1 is that the occurrence of the following predicate in START WITH and CONNECT BY clauses

```
   relation = ‘IS_A’
```

is replaced with the following predicate:

```
   relation != ‘EQV’
```

Note that if a user wants to retrieve all cuisines except Latin American cuisine, then the query can be formulated using the operator `ONT_RELATED` returning 0 as follows:

```
          ……
          ONT_RELATED(sf.cuisine,
                      ‘IS_A’,
                      ‘Latin American’,
                      ‘Cuisine_ontology’)=0;
```

3.2.5 Handling Combination of OR, AND, and NOT. OR and NOT operators are directly specified in the CONNECT BY clause and AND operators are handled by INTERSECT. All conditions are rewritten as conjunctive conditions. For example, `‘A OR (B AND C)’` will be converted into `‘(A OR B) AND (A OR C).’` Then, `‘(A OR B)’` and `‘(A OR C)’` are specified in the CONNECT BY clause in separate queries that can be combined with INTERSECT operator.

3.3 Speeding up ONT_RELATED and ONT_EXPAND Operations

Finding transitive closure from an ontology can be a time-consuming process especially if the ontology has a large number of terms. In addition, different relationship types can further increase the computation cost. To address this problem, a *transitive closure table* is pre-computed. Note that as part of this computation both distance and path measures are computed as well. For the example cuisine ontology, the transitive closure table will be as shown in Table 3.

Table 3: Transitive Closure Table

RootTerm	RelType	Term	Distance	Path
…				
Latin American	IS_A	Mexican	1	…
Latin American	IS_A	Portuguese	1	…
…				

The data is stored in a key compressed index-organized table [18] (primary B^+-tree) with `<RootTerm, RelType, Term>` as the key. The commonly occurring `<RootTerm, RelType>` prefixes are compressed. The distance and path are stored as overflow-resident columns. This allows for basic index-structure to remain compact thereby providing efficient index-lookup.

For a query involving `ONT_EXPAND`, say with arguments `‘Latin American’ and ‘IS_A’` this pre-computed transitive closure table is looked up instead of traversing the ontology to find the transitive closure, and the matching rows are returned. The rows returned include the distance and path measures, which are also available in the Transitive Closure table.

To speed up queries involving `ONT_RELATED`, a new indexing scheme `ONT_INDEXTYPE` is implemented using Oracle’s Extensible Indexing Framework [5]. Users only need to create an index on the column holding ontology terms using `ONT_INDEXTYPE` as follows:

```
CREATE INDEX <index_name>
ON <table_name> (<term_column>)
INDEXTYPE is ONT_INDEXTYPE
PARAMETERS(‘Ontology =
            <ontology_name>’);
```

The basic processing of indexing scheme works as follows. Consider the following index creation statement:

```
CREATE INDEX idx1
ON served_food (cuisine)
INDEXTYPE is ONT_INDEXTYPE
PARAMETERS(‘Ontology=cuisine_ontology’);
```

The index creation results in creation of a key-compressed index-organized table with two columns `<cuisine, row_id>` as shown in Table 4. The `row_id` column contains the row identifier for the `served_food` table.

Table 4: Index Table

cuisine	**row_id**
…	
Mexican	ROWID7
Portuguese	ROWID8
…	

Now, a query involving `ONT_RELATED` operator say with arguments `(sf.cuisine, ‘IS_A’, ‘Latin American’, …)`, is executed by first searching the transitive closure table using the key `(‘Latin American’, ‘IS_A’)` to find the terms, and then for each term the corresponding row identifier is obtained by doing a lookup into the Index Table.

If a query with `ONT_RELATED` operator references `ONT_DISTANCE` and/or `ONT_PATH`, then the indexed implementation of `ONT_RELATED` operator retrieves distance and/or path measures from the transitive closure table. These values are simply returned as part of `ONT_DISTANCE` and `ONT_PATH` invocations.

The index `idx1` created on `served_food` table behaves likes a regular index, which can be incrementally maintained. That is, if a new row is added to `served_food` table, the corresponding `<cuisine, row_id>` values are added to the index table. Similarly, the delete and update operations also result in incremental maintenance of the index.

The transitive closure table is meant for a stable ontology. If the ontology changes, the table needs to be updated. For inserts/deletes/updates into ontology, the transitive closure table can be incrementally maintained. The algorithm is omitted due to lack of space.

3.4 Performance Study using Cancer Ontology

To characterize the performance of ontology-based semantic matching, a part of the National Cancer Institute’s Thesaurus and ontology [17] was stored using a prototype implementation built on Oracle RDBMS. Specifically, the following experiments were conducted using Oracle9*i* Release 2 (9.2.0.3) on a SunOS 5.6, Ultra-60 Sparc Workstation with one 450Mhz CPU and 512 MB of main memory.

The stored ontology consisted of 25,762 terms and 54,387 relationships among them. The resulting transitive closure took 6 minutes to build and contained 186,211 rows.

The following query was executed with the index (transitive closure table) for several different terms. This queries the ontology directly.

```
SELECT count(*) FROM TABLE(
 ONT_EXPAND(NULL, ‘IS_A’,
             :term,
           ‘Cancer_Ontology’));
```

Figure 4 illustrates that query runtime increases linearly as the number of rows in the ONT_EXPAND result set increases.

Next a series of database tables, named `patients`, were created with a varying number of rows, such that 10% of all rows contained a diagnosis term that was one of the 961 subclasses of `‘Experimental_Organism_Diagnoses’ (EOD)`.

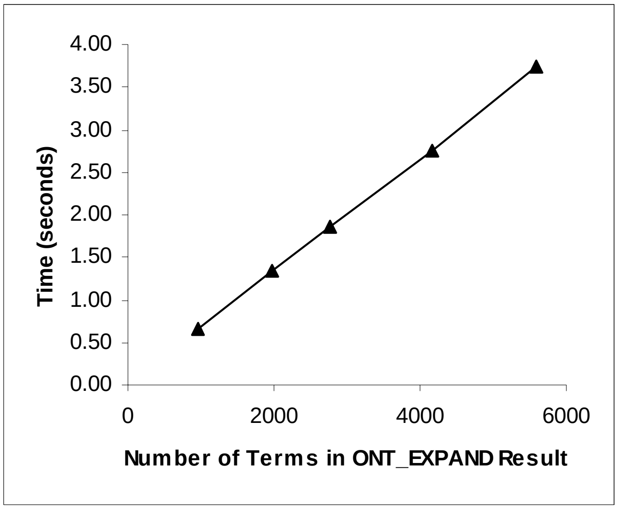


Figure 4: Performance of ONT_EXPAND

Given these patients tables, the following query was executed, using all three ONT_RELATED evaluation mechanisms (functional evaluation without an index, functional evaluation with an index, and index evaluation), to count the number of patients whose diagnosis is an EOD:

```
SELECT count(*) FROM patients
WHERE ONT_RELATED(diagnosis,
                  'IS_A',
     'Experimental_Organism_Diagnoses',
                  'Cancer_Ontology') = 1;
```

Figure 5 illustrates how query runtime varies as the number of rows in the patients table changes.

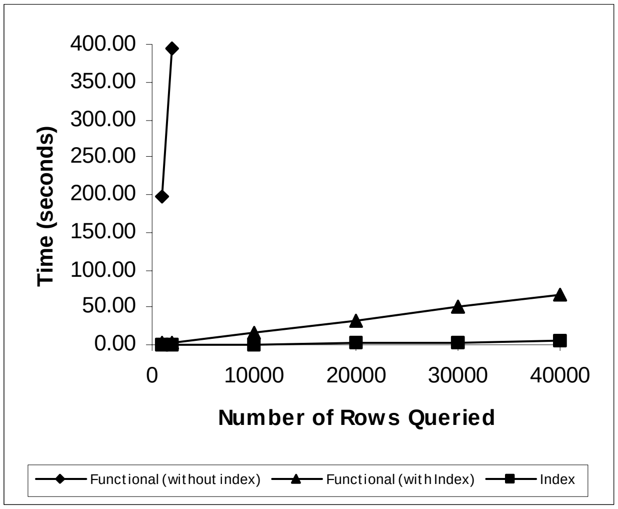


Figure 5: Performance of ONT_RELATED

Functional evaluation without an index computes transitive closure for ONT_RELATED using CONNECT BY clause (without using a pre-computed transitive closure table). Functional evaluation with an index uses the transitive closure table to check if two terms are connected. Index evaluation uses the index table (Table 4) and the transitive closure table to compute the result. Again the query time increases linearly as the result set size increases. The results clearly show that finding transitive closure by traversing the ontology during query processing time is time-consuming and utilizing the pre-computed transitive closure table provides significant performance gains.

4. Ontology-driven Database Applications

This section illustrates the usage of the ontology-related semantic matching operations by considering several database applications.

4.1 A Homeland Security Application

An intelligence analyst for homeland security would be very much interested in an activity that might be related to terrorism. From the example pattern given in [20], where a person rents a truck and another person buys fertilizer, and they reside in the same house, we can formulate an SQL query using ONT_RELATED operator for a given activity table:

Person_name	**Address**	**Activity**	**Object**
John Buck	Addr1	Rent	Ford F-150
Jane Doe	Addr1	Buy	Ammonium Nitrate
…	…	…	…

```
SELECT *
FROM ACTIVITY x, ACTIVITY y
WHERE
 x.Activity = 'Rent' AND
 y.Activity = 'Buy' AND
 ONT_RELATED(x.object,'IS_A','Truck',
            'vehicle_ontology') = 1 AND
 ONT_RELATED(y.object,'IS_A','Fertilizer',
           'chemical_ontology')=1 AND
 x.Address = y.Address;
```

By referring to more than one ontology we can analyze suspicious activities involving a combination of different actions.

4.2 A Supply Chain Application

A supply chain where thousands of products and services are exchanged has a major issue of standardizations of purchase order, bill of material, catalogs, etc. There could be name, language, currency, and unit differences to resolve to name a few. Standardization efforts such as RosettaNet [22], UNSPSC [23] for product category and ebXML [24] to standardize business processes and products are not enough to meet individual vendor/customer needs to resolve semantic differences. Typically, these conflicts have been resolved using some form of mapping mechanisms [6].

These conflicts can be resolved by issuing an SQL query with ONT_RELATED operator using ontologies. Even individually developed ontologies may need the ontology mappings to communicate each other. We can apply ONT_RELATED and ONT_EXPAND operators to the mapping ontology to resolve the conflicts as well.

4.3 A Life Science Application

Life Science domain applications have been using ontologies for representing knowledge as well as the basis of information integration of several heterogeneous sources of life science web databases. Let us consider Gene Ontology (GO)[12], which is primarily used to represent the current knowledge in this domain. It allows user to query using SQL. One sample query is '*Fetching every term that is a transmembrane receptor*'. The GO Graph is stored using the 'term' (=node) and 'term2term' (=arc) tables. It also maintains a table called ‘graph_path’, which stores all paths between a term and all its ancestors. In GO database the following SQL query can be issued for this purpose:

```
SELECT
  rchild.*
FROM
  term AS rchild, term AS ancestor,
  graph_path
WHERE
  graph_path.term2_id = rchild.id and
  graph_path.term1_id = ancestor.id and
  ancestor.name = 'transmembrane
                   receptor';
```

The following query can be used if the data is stored in Oracle RDBMS:

```
SELECT * FROM TABLE
 (ONT_EXPAND (NULL, ‘IS_A’,
             ‘transmembrane receptor’,
             ‘gene_ontology’));
```

The key difference is that the GO database exposes the underlying schema and requires users to formulate queries against those tables, whereas the operator approach attempts to simplify the specification.

4.4 A Web Service Application

Similarly, web service applications can also utilize the ONT_RELATED operator to match two different terms semantically. Consider the web-service matching example described in [8], where user is looking to purchase a sedan. The user requests a web service using the term ‘Sedan’. The vehicle ontology contains ‘Sedan’, ‘SUV’, and ‘Station Wagon’ as subclasses of the term ‘Car’. The web service can alert the user by using the query with the ONT_RELATED operator as follows:

```
    ……
  ONT_RELATED(user_request,
              ‘IS_A’,
              ‘Car’,
              ‘Vehicle_Ontology’) = 1;
```

The degree of match between terms, i.e. how closely the terms are related, as described in [8], can be handled using ONT_DISTANCE and ONT_PATH operators.

5. Experiences

This section summarizes our experience with supporting ontology-related functionality using Oracle RDBMS.

Ontology Access and Manipulation. Instead of providing an API to access and manipulate ontology, we opted for using SQL capabilities. For semantic matching, a set of SQL operators was introduced. The key benefit is that user can fully exploit the expressive power of SQL when performing ontology-based semantic matching

Ontology Storage. In contrast to system like Gene Ontology Database, where a single domain-specific ontology is handled by a set of tables, we decided to store ontologies in a domain-independent manner. That is, multiple ontologies belonging to different domains are stored in a single set of Oracle tables. The key benefit is that a set of canonical operators can be used for semantic match using any ontology. However, the domain-specific approach can lead to a more efficient implementation.

Transitive Closure Table Computation and Its Maintenance. We decided to pre-compute transitive closure table as is done in Gene Ontology Database as well. Their motivation for this pre-computing was to deal with the non-availability of recursive querying capability in the RDBMS used (which is not the case in Oracle). Our primary motivation for pre-computing the transitive closure table was to speed up queries involving semantic matching. The preliminary performance results (as discussed in Section 3.4) clearly demonstrate the significant performance gains by the use of the transitive closure table. The overhead of storing pre-computed transitive closures can be reduced by storing only those closures that are frequently used. The performance gains far outweigh the space overhead drawback.

6. Conclusions and Future work

With the increasing importance of ontologies in business database application areas, it is important that ontologies are stored in databases for efficient maintenance, sharing, and scalability. Equally important is the capability for ontology-based semantic matching in SQL for performance and ease of application development.

The paper addresses these issues by allowing OWL Lite and OWL DL based ontologies to be stored in Oracle RDBMS and by providing a set of SQL operators for ontology-based semantic matching. The approach was further validated by building a prototype implementation on Oracle RDBMS and conducting performance experiments with National Cancer Institute’s Thesaurus and ontology.

In future, we plan to support ontology merging (including semantic matching across ontologies) and ontology evolution. Also, we plan to explore extending the inference engine to support user-defined rules.

Acknowledgments

We thank Jay Banerjee for his comments on an earlier version of this paper.

References

[1] Y. Kalfoglou, J. Domingue, E. Motta, M. Vargas-Vera, and S. Buckingham-Shum, “MyPlanet: an ontology-driven Web-based personalised news service,” In *Proceedings IJCAI 2001 workshop on Ontologies and Information Sharing,* Seattle, WA, Aug. 2001.

[2] F. T. Fonseca, M. J. Egenhofer, "Ontology-Driven Geographic Information Systems," In *Proceedings of the 7th ACM Symposium on Advances in Geographic Information Systems*, pp. 14-19, Nov. 1999.

[3] M. Musen, S. Tu, A. Das, and Y. Shahar, "EON: A component-based architecture for automation of protocol-directed therapy," In *Proceedings of 5th Artificial Intelligence in Medicine in Europe,* pp. 3 -13, Jun. 1995.

[4] T. Berners-Lee, J, Handler, O Lassila. "The Semantic Web," *Scientific American,* May 2001.

[5] J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal, S. DeFazio, "Extensible Indexing: A Framework for Integrating Domain-Specific Indexing into Oracle8*i*," In *Proceedings of the 16th International Conference on Data Engineering,* pp. 91-100, Feb. 2000.

[6] C. A. Knoblock and S. Minton, "Building Agents for Internet-based Supply Chain Integration," *Proceedings of the Workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain*, 1999.

[7] G. van Heijst, A. Th. Schreiber and B. J. Wielinga, "Using Explicit Ontologies for KBS Development," *Int. Journal of Human-Computer Studies,* 46(2-3), pp. 183-292, 1997.

[8] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," *Proceedings of Int. Semantic Web Conference*, pp. 333-347, 2002.

[9] M. Uschold and R. Jasper, "A Framework for Understanding and Classifying Ontology Applications," *Proceedings of IJCAI Workshop on Ontologies and Problem-Solving Methods,* Aug. 1999.

[10] L. D. Stein, "Integrating Biological Databases," *Nature Reviews (Genetics)*, Vol. 4, pp. 337-345, May 2003.

[11] R. Murthy, S. Sundara, N. Agarwal, Y. Hu, T. Chorma, J. Srinivasan, "Supporting Ancillary Values from User Defined Functions in Oracle", In *Proceedings of the 19th International Conference on Data Engineering,* pp. 151-162, 2003.

[12] Gene Ontology Consortium, http://www.geneontology.org.

[13] OntoEdit,
http://www.ontoknowledge.org/tools/ontoedit.shtml.

[14] OntoBroker,
http://ontobroker.semanticweb.org

[15] B. Motik, A. Maedche, and R. Volz, "A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications," In *Proceedings of the 2002 Confederated Int. Conferences DOA/CoopIS/ODBASE*, 2002.

[16] A. Das, W. Wu, and D. McGuinness, "Industrial Strength Ontology Management," *The Emerging Semantic Web*, IOS Press, 2002.

[17] National Cancer Institute Thesaurus,
http://www.mindswap.org/2003/CancerOntology.

[18] J. Srinivasan, S. Das, C. Freiwald, E. I. Chong, M. Jagannath, A. Yalamanchi, R. Krishnan, A. Tran, S. DeFazio, J. Banerjee, "Oracle8*i* Index-Organized Table and its Applications to New Domains," In *Proceedings of the 26th Int. Conf. on Very Large Data Bases,* pp. 285-296, Sept. 2000.

[19] OWL Web Ontology Language Reference,
http://www.w3.org/TR/owl-ref

[20] T. Coffman, S. Greenblatt, and S. Marcus, "Graph-based Technologies for Intelligence Analysis," *Communications of the ACM*, Vol. 47, No.3, pp. 45-47, Mar. 2004.

[21] Jena2 – A Semantic Web Framework,
http://www.hpl.hp.com/semweb/jena.htm

[22] RosettaNet, http://www.**rosettanet**.org/RosettaNet

[23] UNSPSC, http://www.**unspsc**.org

[24] ebXML – Enabling A Global Electronic Market,
http://www.**ebxml**.org/