SWARD: Semantic Web Abridged Relational Databases

Johan Petrini and Tore Risch

Department of Information Technology
Uppsala University
Sweden
{Johan.Petrini,Tore.Risch}@it.uu.se

Abstract

The semantic web represents meta-data as a relation of triples using the RDF data model. We have developed a virtual repository system that enables to process queries to RDF views of large relational databases. This provides very flexible views of wrapped relational databases that can be queried using either the RDF-oriented query language RDQL or SQL. Optimization of queries to such views is challenging because the naïve implementation of such general views become very complex. These complex RDF views make it critical to optimize not only data access time but also the time to perform the query optimization itself. We have developed query optimization techniques based on query rewrites and compile time evaluation of subexpressions to reduce the query size during query processing. This enables execution of real-world queries to RDF views of relational databases.

1. Introduction

The semantic web initiative [1] aims at providing Internet-wide standards for semantically enriching and describing web resources. Using the standards RDF [14] and RDF-Schema (RDFS) [22] any resource can be annotated with *properties* describing its structure and contents. The resources are not limited to web pages but include everything that can be identified through the web. Semantic web applications, for example Dublin Core [10], Open Directory [18], RSS 1.0 [23], use RDF for different kinds of tasks.

RDF repository systems [4][6][7][19][30] offer storage of RDF data and the ability to search RDF data using a query language. However, as most information still resides in relational databases, it is desirable that this information is also exposed to the semantic web through RDF.

The SWARD (Semantic Web Abridged Relational Databases) system provides RDF views of data stored in an existing relational database. General queries are supported over these views. Since RDF views include both schema data and table content data, queries to these views are very flexible and, unlike SQL queries to relational tables, queries can mix meta-data and table access. For example, a query can easily be expressed that given the name of a person find all his properties except the social security number. The challenge is scalability in terms of i) size of database contents, ii) query complexity, and iii) complexity of database schema.

The semantic web represents a database as a collection of *RDF triples*[1] on the format *(s, p, v)* where *s* is called the *subject* (modeling some entity), *p* is called the *predicate* of *s* (modeling some property of an entity), and *v* is called the *object* (the value of *p*). To avoid confusion with ordinary programming terminology we use the terms *property* and *value* instead of *predicate* and *object*. The objects representing *s*, *p*, and *v* are called *RDF resources*. *URIs* [3] (*Universal Resource Identifiers*) uniquely identify RDF resources.

RDF data is usually defined in terms of a predefined set of pre-named properties, called an *ontology*. For example, GovML [28] defines an ontology for eGoverment data.

SWARD presents RDF triples derived from a relational database as a single relation of triples, called the *universal property view*, *UPV*. The UPV is internally defined as a union of *property views*, each representing one exported column in the relational database, called the *content view*, and a *schema view* representing the relational meta-data. The UPV is automatically generated, given that the user specifies for a given relational database and ontology a *property mapping table* that declares how exported relational columns correspond to properties of the used ontology. The user also specifies a *class mapping table* that declares URIs of exported relational tables.

As real-world relational databases often have many columns, queries to the UPV require efficient processing of queries over large unions of many property views. We have evaluated a number of different strategies [19][20] to achieve scalability with respect to i) query execution time

[1] Called *statements* in RDF.

as the database increases, and ii) query optimization time as the size of the query and the size of the schema increases. The latter turns out to be critical for RDF queries to UPVs. The reason is that RDF queries generate many self-joins to the UPV and the UPVs are defined as disjunctions of many property views. Traditional query processing does not scale at all w.r.t. query optimization time; i.e even rather simple queries to relatively small databases cannot be executed efficiently with a conventional commercial database engine [20].

It is particularly important that queries that do not access the database schema but its contents, *database content queries*, are processed efficiently. These are the kinds of queries that are normally used in relational databases and it is essential that they scale. There are also queries that access only relational schema properties e.g. the name of a relational column. Such queries are called *schema queries*. A third kind of queries, *hybrid queries* join schema and content queries.

We have evaluated a number of query reduction techniques [19][20] for scalable processing of the three kinds of queries to UPVs. The database scalability results are verified by performance evaluations using queries to a UPV of a TPC-H based database. In particular, we show substantially faster query processing times by reducing the query size by using a compile time evaluation technique, *partial evaluation* [12], during optimization [20].

As internal query language SWARD uses ObjectLog [16]. It is an object-oriented internal query representation based on Datalog that is very suitable for RDF query transformations. ObjectLog extends Datalog with OIDs and disjunctive expressions. OIDs are needed to represent typed URIs. For simplicity, in this paper only *plain* literals (i.e. strings) are used, so no OIDs are needed in the examples.

As surface query languages we support initially the RDF query languages RDQL [25] and a subset of SQL. We will also support SparQL [27] [5]. Our approach applies to other proposed semantic web query languages (e.g. [13] [26]) as well.

2. Related Work

RDF repository systems [4][6][7][19][30] often use relational databases internally. Such a relational database is fully managed by the repository system and the schema of the relational database is internal. If one wants to make RDF queries to an existing relational database using such a repository, it requires downloading the database into the repository. This clearly does not scale.

Rather than storing RDF data in dedicated RDF-repositories our work wraps an existing relational database so that it can be used in RDF queries without downloading database tables to a repository. Instead the data necessary for answering a particular query are represented as RDF triples streamed through the wrapper.

SWIM [7] and D2RQ [2] provide conversion methods from relational databases to RDF. None of the works study how to optimize queries over RDF views of relational databases.

The typed RDFS-based view specification language RVL [17] is proposed for semantic web integration [7]. It can complement SWARD by allowing the definition of RDF views on top of our UPVs.

The reference relation by [15] proposes a flexible representation of a relational database as a four-column table. This enables very general queries combining schema and data. Our UPVs provide the same flexibility and, in addition, support RDF mappings.

Optimizing disjunctive queries in general was studied by, e.g., [8] without paying attention to query optimization time and RDF.

Partial evaluation [12] is used very little for query processing and was never used for optimizing large disjunctive queries as UPVs.

To summarize, we are not aware on any other work on optimizing query processing of very large disjunctive queries as is needed for RDF views of relational databases. In particular, the partial evaluation allows very general and scalable queries to UPV of large relational databases.

3. Example

To illustrate our approach we use an example database containing life event data stored in a backend relational database, *eGov*, accessed through RDQL or SQL. The database is queried in terms of the GovML ontology [28].

The following SWARD statement automatically generates the UPV for the exported tables:

```
ExportRDB('JDBC:..;DatabaseName=eGov',
          'eGov',
          'udbl.it.uu.se/schemas/eGov')
```

The first argument to *ExportRDB* is the JDBC connection URL for the relational database, the second is the name of the UPV, and the third is the ontology used by the UPV.

Lifeevent	**Eid**	**Lang**	**Descr**	**Law**	**Form**
	'ABC1234H'	'EN'	'Getting married'	'FRC-234'	http://www.eGov.org/marriage.html

Faqlists	**Eid**	**Faq#**	**Question**	**Answer**
	'ABC1234H'	1	'Possibillity to get married online'	'Yes'

Relatedservices	Eid	Service#	Title
	'ABC1234H'	1	'Issuing a birth certificate'
	'ABC1234H'	2	'Online payment'

Figure 1: Example database.

In addition *ExportRDB* requires a user-defined *property mapping table* (Table 1) stored in SWARD to map 1:1 between exported columns from the relational database and corresponding URIs representing ontology properties, *property identifiers*.[2] Here we show only the mappings for the *Lifeevent* table.

Table 1: Property mapping table.

Table	Column	Ontology	PropID
Lifeevent	Eid	dc:	dc:Identifier
Lifeevent	Lang	govml:	govml:#Language
Lifeevent	Descr	govml:	govml:#Subject
Lifeevent	Eid	egov:	dc:Identifier
Lifeevent	Lang	egov:	govml:#Language
Lifeevent	Descr	egov:	govml:#Subject
Lifeevent	Law	egov:	egov:#Law
Lifeevent	Form	egov:	egov:#Form

Neither *dc:* nor *govml:* include all propertied needed by eGov. Therefore we develop our own ontology *egov:* that extends *dc:* and *govml:* to provide complete mappings for eGov.

To enable representation of the schema view the user must also provide a simple *class mapping table*, *cMap*, (Table 2) that maps, for a given ontology, relational table names to *class identifiers*. The schema view part of the UPV encodes relationships between RDF-Schema *classes* and *properties*, and relational database schema elements. Class identifiers are represented as URIs of the standard RDF-Schema class *rdfs:Class* while the property identifiers belongs to the class *rdf:Property*. The class for which a property is defined is called the *domain* of the property and represented by the RDF-Schema property *rdfs:domain*. The value of a property also belongs to some RDF-Schema class, called its *range*. The range of a property is represented by the RDF-Schema property *rdfs:range*. The range currently always is the general RDF-Schema class *rdfs:Literal*. Class memberships of URIs are specified by the property *rdf:type*. Here we show only the mappings for the *Lifeevent* table.

[2] *govml:* is namespace for the ontology *http://www.egov_project.org/GovMLSchema* , *dc:* is namespace for the ontology *http://purl.org/dc/elements/1.1/* and *egov:* for the ontology *udbl.it.uu.se/schemas/eGov*

Table 2: Class mapping table.

Table	Ontology	ClassID
Lifeevent	egov:	egov:#LifeEvent

There are several other classes and properties in the RDF-Schema model which are not needed for the mapping. Notice that the user has to specify only the class and property mapping tables; the schema view is automatically inferred from these tables.

With the above property and class mapping tables the single row in table *Lifeevent* will produce the UPV named *eGov* in Table 3. Section 4 explains the rules for how the definition of *eGov* is automatically generated from the property and class mapping tables.

Table 3: Universal property view for part of Lifeevent table (i.e. the *Descr* column).

eGov	S	P	V
	dc:Identifier/ ABC1234H	govml:#Subject	'Getting married'
	egov:#LifeEvent	rdf:type	rdfs:Class
	Govml:#Subject	rdf:type	rdf:Property
	Govml:#Subject	rdfs:domain	egov:#LifeEvent
	Govml:#Subject	rdfs:range	rdfs:Literal

Here '*dc:Identifer/ABC1234H'* is a system generated URI that identifies a life event by concatenating the property identifier for the key column in table *Lifeevent* , *dc:Identifier*, with the key value *ABC1234H*. The string 'Getting married' is the value of the column *Subject* for that row.

The example RDQL content query to the UPV in Figure 2 returns all life event forms about marriage.

```
SELECT ?val2
FROM <udbl.it.uu.se/upv/egov/>
WHERE
      (?s,<govml:#Subject>,?val1),
      (?s,<egov:#Form>,?val2)
AND   ?val1 =~ '%married%'
```

Figure 2: Example RDQL query.

In RDQL URIs are specified on the form <…> and variables are prefixed with '?'. The SELECT clause specifies the tuples to be returned to the user. The FROM clause specifies the source to query. Before querying a UPV from RDQL the user must specify a URI acting as an alias for the UPV name. Here the UPV, *eGov*, is accessible from RDQL by the URI *udbl.it.uu.se/upv/egov/*. The WHERE clause specifies a selection condition over the RDF triples in the UPV. The

selections are specified using the notation *(s,p,v)* where *s* (subject), *p* (property), and *v* (value) are constants or variables and filters.

The result from the queries is the tuple[3]:

```
('http://www.eGov.org/marriage.html')
```

The example query is expressed in SQL as in Figure 3. Notice that SQL requires many self joins making the query less natural for querying RDF than the corresponding RDQL query. The reason is SQL's reliance on tuple calculus, while RDQL is based on domain calculus. SWARD supports both query languages, though.

```
SELECT t2.value
FROM eGov AS t1, eGov AS t2
WHERE
  t1.property = 'govml:#Subject'  AND
  t1.subject = t2.subject         AND
  t2.property = 'egov:#Form'      AND
  t1.value LIKE '%married%'
```

Figure 3: Example SQL query.

The FROM clause in the SQL query specifies an identifier for the UPV to query.

4. Universal property views

The parser first translates the query to the ObjectLog expression in Figure 4 querying the UPV *eGov*.

```
1. {val2 |
2.  eGov(s,'govml:#Subject',val1)  AND
3.  eGov(s,'egov:#Form',val2)      AND
3.  like(val1,'%marriage%')}
```

Figure 4: ObjectLog expression of example RDQL query.

This section outlines how UPVs are automatically defined by *ExportRDB*.

4.2 Defining universal property views for relational databases

A query references a UPVs in the FROM clause. The query processor uses the UPV definition and the back-end relational database schema to translate the query into an algebra expression containing one or several calls to SQL in the wrapped relational database. The UPV *U* of a relational database for a given ontology is defined as the union of two subviews, one representing the schema of the relational database, the *schema view S*, and one representing its contents, the *content view C*, i.e. $U=S \cup C$. *U* is generated by *ExportRDB* and has the definition

```
U(s,p,v) :- S(s,p,v) OR C(s,p,v)
```

In our example *U* is named *eGov*, *S* is S_{eGov} and *C* is C_{eGov}. Given an ontology, the schema view *S* maps RDF-Schema classes to tables and properties to columns. The *class view*, *Classes(s,p,v)*, specifies how relational table names are mapped to RDF-Schema class identifiers. The *domain view*, *Domains(s,p,v)* specifies for every property identifier mapped to a column the class identifier of its table. The *range view*, *Ranges(s,p,v)* specifies the type of a relational column as an RDF-Schema class identifier. These three views are sufficient to map any relational database table to RDF-Schema. It holds that:

```
S(s,p,v) :- Classes(s,p,v) OR
            Domains(s,p,v) OR
            Ranges(s,p,v)
```

The content view *C* of a relational database for an ontology is defined as a union of internal *property views* PV_a generated for each exported column *a* in the database, i.e. $C = PV_a$. Figure 5 shows the generated definition of *U* for our example with *C* expanded on lines 3-6. Notice that real-world relational databases contain many columns so the disjunctive expression will be large. The schema view is called on line 2.

```
1. U(s, p, v):-
2.  S(s, p, v)                    OR
3.  Eid(s, p, v)                  OR
4.  Lang(s, p, v)                 OR
5.  Descr(s, p, v)                OR
6.  Law(s, p, v)                  OR
7.  Form(s, p, v)
```

Figure 5: Universal property view definition for *Lifeevent* table.

Figure 6 shows the definition of the property view *Descr(s, p, v)*.

```
1. Descr(s, p, v):-
2. lifeevent(eid, lang, v, law)          AND
3. pMap('Lifeevent','Eid','egov:',kpid)AND
4. s = rowid(kpid,eid)                   AND
5. pMap('Lifeevent','Descr','egov:',p) AND
```

Figure 6: Property view for *Descr* column

Line 2 accesses the relational table *Lifeevent*. Line 3 accesses the property mapping table to get the property identifier *kpid*, given table *Lifeevent*, column *Eid*, and the ontology *egov:*. The function *rowid* in line 4 takes as arguments the property identifier *kpid* for the key column in *Lifeevent* and a key *eid*. It returns a unique table row identifier by string concatenation, e.g. '*dc:Identifier/ABC124H*'. Line 5 retrieves the property identifier for column *Descr*. In the definition of *Descr* partial evaluation is used to eliminate the two calls to *pMap* on lines 3 and 4; the query processor accesses *pMap* during optimization to get binding for variables *kpid* and *p*. This reduces the size of the body of *Descr* with 50%.

The ObjectLog query in Figure 4 contains only two references to the UPV, *eGov*, (lines (2-3)). However, most real-world queries will contain many self-joins and this will make the expanded expression huge. A challenge is

[3] We use the (..) notation for tuples.

therefore to investigate query reduction strategies to handle this complexity such as partial evaluation.

5. Overview of SWARD query processing

Figure 7 illustrates the query processor of SWARD. Applications access SWARD through its *query interface*. When a user executes a query it is first transformed by the *parser* into a ObjectLog expression, e.g. the query in Figure 4.

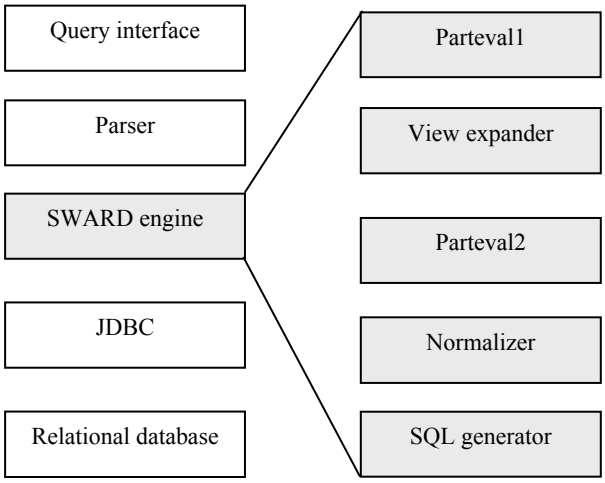

Figure 7: System architecture.

The steps *parteval1* and *parteval2* perform query reduction by partial evaluation of query expressions [19] [20]. The *view expander* substitutes each reference to the UPV in the query with its definition. In our example this first produces the expression illustrated by Figure 8. Then the schema views *S* and content views *C* and their subviews are expanded recursively. The final expanded view may become very large which slows down query optimization (remember that each content view is defined as a union of property views) Therefore we use partial evaluation to reduce the size of the query before view expansion.

The *normalizer* transforms the simplified query to disjunctive normal form. Normalization improves query execution by combining in the same conjunctive subqueries predicates from the query and predicates from property view definitions. However, normalization is very expensive for large expressions. Partial evaluation before normalization is thus very important and the view expanded expression is substantially reduced before normalization by *parteval2* [20].

```
1.{val2 |
2.(S(s,'govml:#Subject',val1)    OR
3. C(s,'govml:#Subject',val1))          AND
4.(S(s,'egov:#Form',val2)        OR
5. C(s,'egov:#Form',val2))              AND
3.like(val1,'%marriage%')}
```

Figure 8: Example query after expanding the universal property view.

After normalization the SQL generator finally translates each simplified conjunctive subquery into an algebra expression as in [11]. The algebra expression contains calls to a foreign function sending SQL statements to the *relational database* via *JDBC*. The only SQL statement submitted to the back-end relational database in our example is:

```
SELECT form
FROM lifeevent
WHERE descr LIKE '%married%'
```

The algebra expression further contains some calls to *rowid* to manage the construction of row identifiers. The algebra expression is finally interpreted.

7. Summary and conclusions

We are developing a system named SWARD for scalable access in terms of an ontology to large relational databases. Our approach is to view the information, i.e. schema and content, of a relational database as a large disjunctive *universal property view (UPV)* defined using automatically generated expressions in a Datalog dialect called ObjectLog [16].

The UPV is automatically generated, given that the user specifies for a given relational database and ontology a *property mapping table* that declares how exported relational columns correspond to property identifiers of the used ontology and a *class mapping table* that declares how relational tables correspond to class identifiers of the used ontology.

RDF queries expressed in RDQL or SQL are translated into ObjectLog queries over the UPV. The UPV is internally defined in ObjectLog as a disjunction of property views, each representing one exported column in the relational database, and a schema view representing the relational meta-data. Partial evaluation reduces the ObjectLog expressions to improve query processing time. An SQL generation finally generates algebra expressions containing SQL calls to the back-end relational database.

Future work includes investigating query transformation techniques for mediation of data from different sources [24] accessible through web services. Mediators are actually view definitions combining and reconciling data from different sources.

References

1. T.Berners-Lee, J.Hendler, and O.Lassila: The Semantic Web, *Scientific American*, May 2001.
2. C.Bizer, A.Seaborne: D2RQ -Treating Non-RDF Databases as Virtual RDF Graphs (Poster). *3rd*

International Semantic Web Conference (ISWC2004), 2004, Hiroshima, Japan.
3. D.Brickley and R.V.Guha: RDF Vocabulary Description Language 1.0: RDF-Schema, http://www.w3.org/TR/rdf-schema/, 2004.
4. J. Broekstra, A. Kampman and F. van Harmelen: Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema. *Proc.1st International Semantic Web Conference (ISWC'02)*, Sardinia, Italy. 2002.
5. Y.Cao: Processing SparQL Queries in an Object-Oriented Mediator, Uppsala Master's Theses in Computer Science 306, http://user.it.uu.se/~udbl/Theses/YuCaoMSc.pdf.
6. E.I.Chong, S.Das, G.Eadon and J,Srinivasan: An Efficient SQL-based RDF Querying Scheme, *Proc. 31st Intl. Conf. on Very Large Databases, VLDB2005,* pp1216-1227, Trondheim, Norway, 2005
7. V.Christophides, G.Karvounarakis, A.Magkanaraki, D.Plexousakis, and V.Tannen: The ICS-FORTH Semantic Web Integration Middleware (SWIM), *IEEE Data Engineering Bulletine*, 26(4), Dec. 2003.
8. J.Claußen, A.Kemper, G.Moerkotte, K.Peithner, and M.Steinbrunn: Optimization and Evaluation of Disjunctive Queries. *IEEE Trans. Knowl. Data Eng.* 12(2): 238-260 (2000)
9. C.Consel, L.Hornof, J.L.Lawall, R.Marlet, G.Muller, J.Noyé, S.Thibault, and E-N.Volanschi. "Tempo: Specializing Systems Applications and Beyond". *ACM Computing Surveys*, 30(3), September 1998.
10. Dublin Core Meta-data Initiative, Dublin Core Metadata Element Set, V 1.1, http://dublincore.org/documents/dces/
11. G. Fahl and T. Risch: Query Processing over Object Views of Relational Data, *The VLDB Journal*, 6(4): 261-281, 1997.
12. N. D. Jones. An Introduction to Partial Evaluation. *ACM Computing Surveys*, 28(3), 1996.
13. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis and M. Scholl: RQL: A Declarative Query Language for RDF, *Proc. International World Wide Web Conference (WWW'02)*, Honolulu, Hawaii, USA. 2002.
14. G.Klyne and J.J.Carroll: Resource Description Framework (RDF): Concepts and Abstract Syntax, http://www.w3.org/TR/rdf-concepts/, 2004.
15. W.Litwin, M.Ketabchi, R.Krishnamurthy: First Order Normal Form for Relational Databases and Multi Databases, *SIGMOD Records*, 20(4), December 1991.
16. W. Litwin, and T. Risch, Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering* , 4(6), 1992, pp. 517-528.
17. A.Magkanaraki, V.Tannen, V.Christophides, and D.Plexousakis: Viewing the Semantic Web Through RVL Lenses, *2nd International Semantic Web Conference (ISWC'03)*, 2003, Sanibel Island, Florida, USA.
18. Open Directory Project, http://dmoz.org/
19. J.Petrini and T.Risch: Processing queries over RDF views of wrapped relational databases, in *Proc. 1st International Workshop on Wrapper Techniques for Legacy Systems,* WRAP 2004, Delft, Holland, November 2004, http://user.it.uu.se/~udbl/publ/WRAP04.pdf.
20. J.Petrini and T.Risch: *Scalable RDF Views of Relational Databases through Partial Evaluation*, Technical Report 2006-016, Dept. of Information Technology, Uppsala University, Sweden, March 2006,http://www.it.uu.se/research/publications/reports/2006-016/
21. RDF Gateway - a platform for the semantic web, Intellidimension, http://www.intellidimension.com/.
22. RDF Vocabulary Description Language 1.0: RDF-Schema, http://www.w3.org/TR/rdf-schema/, 2004.
23. RDF Site Summary 1.0, http://web.resource.org/rss/1.0/
24. T.Risch, V.Josifovski, and T.Katchaounov, Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis (eds.): *Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data,* ISBN 3-540-00375-4,Springer, 2003, pp 211-238.
25. A.Seaborne: RDQL - A Query Language for RDF, W3C Member Submission, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/, 2004.
26. SeRQL, http://www.openrdf.org/doc/sesame/users/ch06.html
27. SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005, http://www.w3.org/TR/rdf-sparql-query/
28. E. Tambouris, G.Kavadias, and E.Spanos: The Government Markup Language (GovML), Journal of E.Government, 1(2), Haworth Press, 2004, http://www.haworthpress.com/web/JEG/.
29. TPC-H benchmark, http://www.tpc.org/tpch/
30. K.Wilkinson, C.Sayers, H.A.Kuno, and D.Reynolds: Efficient RDF Storage and Retrieval in Jena 2, *Proc. VLDB Workshop on Semantic Web and Databases (SWDB'03)*, pp 131-150, September 2003.