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

We have developed a system that can process queries to RDF views of large relational databases. This provides very flexible views of wrapped databases that can be queried using either RDQL or SQL. Query processing over such views is challenging because their naive implementation becomes very complex. This makes it critical to optimize not only data access time but also the time to perform the query optimization itself. We have developed query processing techniques based on partial evaluation of query expressions as a way to enable execution of real-world queries to RDF views of relational databases.

1. Introduction

RDF repository systems [2][4][20] 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 existing relational databases. 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 department finds all its properties except its internal identifier.

RDF data is usually defined in terms of an ontology. For example, GovML [19] 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 a *content view* that represents relational table contents and a *schema view* that represents the relational schema. The content view is defined as the union of *property views*, each representing one exported column in the relational database. 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 RDF-Schema class URIs corresponding to 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. The reason is that RDF queries generate many self-joins to the UPV and the UPV is defined as a large disjunction. Traditional query processing does not scale at all w.r.t. query optimization time (time spent in rewrites and cost-based optimization); i.e. even rather simple queries to UPVs over relatively small databases cannot be executed efficiently with a conventional commercial database engine [14].

It is particularly important that RDF queries accessing database tables, *content queries*, scale. These are the kinds of queries that are normally used in relational databases. We have developed methods for scalable processing of conjunctive content queries to UPVs of relational databases [13] [14]. 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. The methods developed for scalable processing of conjunctive content queries are also applicable to schema and hybrid queries.

As internal query language SWARD uses ObjectLog [11]. It is an object-oriented internal query representation based on Datalog that is very suitable for RDF query transformations. ObjectLog extends Datalog with OIDs, disjunctive expressions and foreign predicates. OIDs are needed to represent typed literals and for distinguishing between URIs and literals. For simplicity, in this paper we represent URIs and literals

as strings and we assume that no string representation of a resource can be both a URI and a literal. Hence no OIDs are needed in the examples.

As user query languages we support initially RDQL [15] and a subset of SQL. We will also support SparQL [18] [3]. Our approach applies to other proposed semantic web query languages (e.g. [9] [17]) as well.

2. Related Work

RDF repository systems [2][4][20] 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 SWARD.

SWIM [6] and D2RQ [1] provide conversion methods from relational databases to RDF, without discussing how to optimize queries over RDF views of relational databases.

The typed RDFS-based view specification language RVL [12] is proposed for semantic web integration [6]. It can complement SWARD by allowing the definition of RDF views on top of our UPVs.

The reference relation by [10] 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., [5] without paying attention to query optimization time and RDF.

To summarize, we are not aware of any other system that offers querying facilities over large disjunctive RDF views of relational databases. An enabling technology we use to achieve this is a compile time evaluation technique, *partial evaluation* [8], of property view definitions to substantially reduce query size [14].

3. Example

To illustrate our approach we use an example database containing life event data stored in a back-end relational database, named *eGovern*, accessed through RDQL or SQL. The schema for *eGovern* is shown in Figure 1. The database is queried in terms of the GovML ontology [19].

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

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

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.

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

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, called *property identifiers*.[1] Here we show 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 properties needed by the UPV *eGov* Therefore we develop our own ontology *egov:* that extends *dc:* and *govml:* to provide complete mappings for the *Lifeevent* table.

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 specifies relational database schema elements as RDF-Schema *classes* and *properties*. Here we show the mappings for the *Lifeevent* table.

[1] *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 http://*udbl.it.uu.se/schemas/eGovern*#

Table 2: Class mapping table.

Table	Ontology	ClassID
Lifeevent	egov:	egov:LifeEvent

Notice that the user has to specify only the class and property mapping tables; both the content and schema view are automatically inferred from these tables.

With the above property and class mapping tables the UPV named *eGov* will, given the single row in table *Lifeevent*, produce a number of RDF triples where some of them are shown 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 *Descr* for that row.

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

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

Figure 2: Example RDQL 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 *http://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. Filters can also be defined.

The result from the query is the tuple[2]:

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

The example query can also be expressed in SQL as in Figure 3. Notice that SQL requires many self joins making it 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 (se [7] for a short description). SWARD supports both query languages, though.

```
SELECT t2.val
FROM eGov AS t1, eGov AS t2
WHERE
  t1.p = 'govml:Subject'     AND
  t1.s = t2.s                AND
  t2.p = 'egov:Form'         AND
  t1.val 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

A query to a UPV is first translated to ObjectLog by the parser. In our example the query to the UPV *eGov* is translated to the ObjectLog expression in Figure 4.

```
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.

The query processor uses the UPV definition to translate the query into an algebra expression containing one or several calls to SQL in the back-end 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}. Three views are sufficient to map any relational database table, given an ontology, to an RDF-Schema. It holds that:

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

The *class view*, *Classes(s,p,v)*, defines the class and property identifiers, representing relational tables and columns respectively, as RDF-Schema classes and properties. The *domain view*, *Domains(s,p,v)* specifies for every property identifier mapped to a column the

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

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.

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. $C = \bigcup_a PV_a$. Figure 5 shows the generated definition of UPV *eGov* for our example with C_{eGov} view expanded on lines 3-7. 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. eGov(s, p, v):-
2.   SeGov(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, form) AND
3. Map('Lifeevent','Eid','egov:',kpid)AND
4. rowid(kpid,eid,s)                  AND
5. pMap('Lifeevent','Descr','egov:',p)
```

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 foreign predicate *rowid* in line 4 takes as arguments the property identifier *kpid* for the key column in *Lifeevent* and a key *eid*. It generates a unique table row identifier by string concatenation, e.g. '*dc:Identifier/ABC124H*'. Line 5 retrieves the property identifier for column *Descr*.

Notice that the expression 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 therefore to investigate query processing strategies to handle this complexity.

5. Overview of SWARD query processing

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

The steps *parteval1* and *parteval2* in *rewriter* perform *partial evaluation*, i.e. compile time evaluation of query expressions used in property view definitions to substantially reduce the size of the query [14]. For example, in Figure 6 lines 3 and 5 could be eliminated by partial evaluation as the query processor looks up the *pMap* table, which reduces view *Descr* with two clauses. Similar reductions by partial evaluation substantially improve query processing time [14]. 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 and content views are expanded.

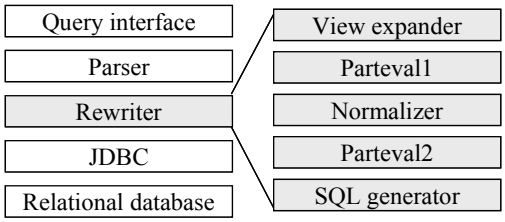

Figure 7: System architecture.

The *normalizer* transforms the simplified query to disjunctive normal form (i.e., a union of conjunctive subqueries). Normalization improves query execution by combining in the same conjunctive subqueries predicates from the query and predicates from property view definitions. However, in this case, normalization produces unreasonable large expressions. Minimizing the query before normalization is thus very important Therefore the view expanded expression is first simplified by *parteval1*.

```
1.{val2 |
2.(SeGov(s,'govml:Subject',val1)    OR
3. CeGov (s,'govml:Subject',val1))      AND
4.(SeGov (s,'egov:Form',val2)       OR
5. CeGov (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. The algebra expression contains calls to SQL statements sent via *JDBC* to the *relational database* for cost-based optimization and execution. The only SQL statement submitted to the back-end relational database in our example is actually [14]:

```
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.

6. Summary and conclusions

SWARD allows scalable access to large relational databases [13] [14]. Both schema and content, of a relational database, is viewed as a large disjunctive *universal property view (UPV)* defined using automatically generated expressions in a Datalog dialect called ObjectLog [11].

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. The SWARD rewriter simplifies and transforms the ObjectLog expressions into algebra expressions containing SQL calls to the back-end repository.

Future work includes investigating query transformation techniques for mediation of data from different sources [16] accessible through web services. Mediators are actually view definitions combining and reconciling data from different sources.

References

1. C.Bizer, A.Seaborne: D2RQ -Treating Non-RDF Databases as Virtual RDF Graphs (Poster). *3rd International Semantic Web Conference (ISWC2004)*, 2004, Hiroshima, Japan.
2. 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.
3. 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.
4. 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
5. 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)
6. 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.
7. R.Elmasri, S.B.Navathe: Fundamentals of Database Systems. Addison-Wesley, 5th edition, 2007.
8. N. D. Jones. An Introduction to Partial Evaluation. *ACM Computing Surveys*, 28(3), 1996.
9. 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.
10. W.Litwin, M.Ketabchi, R.Krishnamurthy: First Order Normal Form for Relational Databases and Multi Databases, *SIGMOD Records*, 20(4), December 1991.
11. 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.
12. 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.
13. 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.
14. 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/
15. A.Seaborne: RDQL - A Query Language for RDF, W3C Member Submission, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/, 2004.
16. 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.
17. SeRQL, http://www.openrdf.org/doc/sesame/users/ch06.html
18. SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005, http://www.w3.org/TR/rdf-sparql-query/.
19. 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/.
20. 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.