Processing Queries over RDF Views of Wrapped Relational Databases

Johan Petrini and Tore Risch

Department of Information Technology, Uppsala University, 75195 Uppsala, Sweden
{Johan.Petrini, Tore.Risch}@it.uu.se

Abstract. The semantic web standard RDF enables web resources to be annotated with properties describing structure and contents. However, there is a vast amount of additional high quality information in the *hidden web* stored in databases accessible from the web but not as web pages. In particular, most organizations use relational database technology and such databases should also be accessible from semantic web tools. We are developing a system to transparently wrap relational databases as *virtual* RDF resource descriptions. The wrapper provides relational database access to the Edutella infrastructure for searching educational resources. Semantic web queries are expressed using a Datalog-based RDF query language that through the wrapper transparently retrieves information from relational database servers. Since the data volume in these databases is huge, the data cannot be downloaded but instead virtual RDF resource descriptions are returned as query results. Query optimization techniques permit transparent and efficient mapping from RDF queries to relational database queries. Semantic web queries are more dynamic than relational database queries and they may freely mix access to data and schema. This makes it necessary to optimize not only data access time but also the time to perform the query optimization itself.

1 Introduction

Modern information systems often need to access many different kinds of data, including Internet-based web resources, relational databases, or files containing experimental results. It is getting increasingly difficult to get the correct information when retrieving information from web resources using traditional unstructured free-text based search methods as provided by, e.g., GOOGLE.

Normally the useful information is hidden inside huge amounts of irrelevant information. This data retrieval problem gets even worse if one wants to combine web resources with other kinds of data stored outside the web, for example in enterprise databases, often referred to as the *hidden web*. Either one has to manually browse-cut-and-paste between web search tools and database tools, or one has to develop hard-wired programs accessing data from relational databases and web sources for combining and filtering the retrieved resources.

The semantic web initiative [3] aims at providing Internet-wide standards for semantically enriching and describing web data. Using the standards RDF [7][14] and

RDF-Schema [4], abbreviated as RDFS, any web resource can be annotated with *properties* describing its structure and contents. This facilitates guided search of web resources in terms of these properties. The properties are represented as sets of RDF *statements*, which are triples containing a web resource (the *subject*), a property (the *predicate*), and a value (the *object*). RDFS [4] adds semantics to basic RDF with schema definition capabilities providing, e.g. classes, inheritance, and restrictions on the kinds of properties a given class of web resources can have. RDF is used, e.g. by the Dublin Core standard [8] for meta-data markup of library data and the Edutella infrastructure [18] uses it for searching educational web resources.

Queries to semantic web data are specified using some of the query languages proposed for this, e.g. RDQL [23], RQL [13], and QEL [21].

We are developing a system SWARD (Semantic Web Abridged Relational Databases) for scalable RDF based wrapping of existing relational databases in the *hidden web*. Instead of downloading the relational database tables into RDF repositories we map an existing relational database schema into a corresponding *virtual* RDF statement set of the wrapped relational database. When semantic web queries reference this virtual statement set the system automatically translates fragments of the queries into one or several SQL queries to the relational database. The result of a query is not explicitly stored as RDF statements in a repository, but statements are instead dynamically generated as data is retrieved from the relational database. Query filters in RDF queries are moved into SQL selections when possible. Filters not expressible in SQL are applied on the results from the SQL queries. A particular problem is that queries to RDF statement sets do not need to distinguish between what is schema and what is data as in relational databases. In RDF both schema and data are mixed in the statement sets and, unlike SQL, queries to RDF sources do not need to be expressed in terms of a database schema. This prohibits pre-compilation of queries as in SQL.

The approach is evaluated initially in the context of the Edutella framework [18], which is a peer-to-peer infrastructure for semantic web searches of educational resources. Edutella uses the query language QEL [21], a Datalog [27] based query language for RDF. Each educational source made available to Edutella is called an Edutella *provider*. A provider receives dynamic QEL queries from Edutella to a specific source. It evaluates the query and returns the result as a set of RDF statements. Our provider permits QEL queries to any wrapped relational database. As test case we provide RDF query access to a relational database storing information about Swedish museums, *Museifönstret*[1][17]. Since, unlike SQL, Edutella queries are always dynamic and cannot be precompiled, we optimize not only the query execution time as relational databases but also the query compilation time as is the focus of this paper.

Our approach enables efficient semantic web peer-to-peer queries to the combination of resources on the web and in the *hidden web*. It allows Edutella peers to access existing relational databases as well as other sources, even though the relational databases have totally different data representations than RDF-statements. The system manages mappings between meta-data descriptions based on Dublin Core used in the Edutella ontology and the wrapped relational databases. The system furthermore allows user functions implementing algorithms for data filtration, indexing, and fusion, and it enables transparent use of these user-defined functions in queries and views.

[1] English:'The Window to Museums'.

2 Related Work

Usually, RDF statements are stored as web documents or in internal relational databases designed for RDF [2][26]. The schema of the relational database is internal to the repository system. One problem with storing all data as triples is that the repository does not have any knowledge about the most efficient representation for each application, which makes relational query optimization less effective. To alleviate this, Jena2 [26] uses property tables for non-triple representation of RDF statements.

However, if one wants to access existing large relational databases through semantic web queries using an RDF repository one needs to download the relational database tables into the RDF repository before querying them. This can be very costly if the relational database is large. By contrast regular relational databases are designed and tuned for maximal efficiency of the applications using the database. To limit data transmission they are designed for keeping all data in the database and only export a minimal amount of data for answering queries. The database schema provides application-oriented meta-data descriptions, efficient data representation, and efficient query processing.

Rather than storing RDF data in dedicated RDF-repositories our work wraps existing relational databases to be used in the semantic web queries without downloading database tables to a repository. Instead the statements necessary for answering a particular query are represented as virtual statements streamed through the wrapper. The closest works are D2R MAP [5], and RDF Gateway [20], which provides conversion methods from relational databases to RDF. The typed RDFS-based view specification language RVL [16] is proposed for semantic web integration [6]. However, none of the works deal with how to actually optimize semantic web queries over wrapped relational databases, the main topic of this work.

Several mediator projects [10][11][12][15][19][22][25] wrap external data sources from virtual databases. However, none of these projects deal with wrapping relational databases under semantic web infrastructures.

There are a few proposals for query languages for RDF e.g. RDQL [23], RQL [13], and QEL [21]. These query languages are based on declarative queries to the space of triples constituting an RDF database. In this project we primarily use QEL but the technique can be applied on the other query languages as well.

SWARD generalizes the Edutella peer-to-peer infrastructure [18] for searching learning materials on the web to permit providers to execute QEL queries over the *hidden web*.

3 Example

The relational database *Museifönstret* [17], abbreviated as *WM,* stores data about artifacts in Swedish museums. For example, a relation `Resource` in *WM* contains information about artifacts such as for example their name, description, URI and ID according to the schema:

```
Resource(RID,MID,Name,URI,ShortDesc,Desc)
```

RID is a numeric resource identifier and MID is a numeric museum identifier. Our Edutella provider SWARD wraps *WM* to appear as a set of RDF statements.

An example QEL query, *q1*, submitted to SWARD from Edutella is to find all museum artifacts with a name that contains the string 'Matter'. It is expressed in QEL as[2]:

```
@prefix qel:http://www.edutella.org/qel#
@prefix dc:http://purl.org/dc/elements/1.1/
?(x,t):-qel:s(x,'dc:title',t),
        qel:like(t,'Matter')
```

Here we use a Datalog-like syntax for the QEL query [21]. In practice it is sent from Edutella to the provider using a less readable equivalent XML syntax.

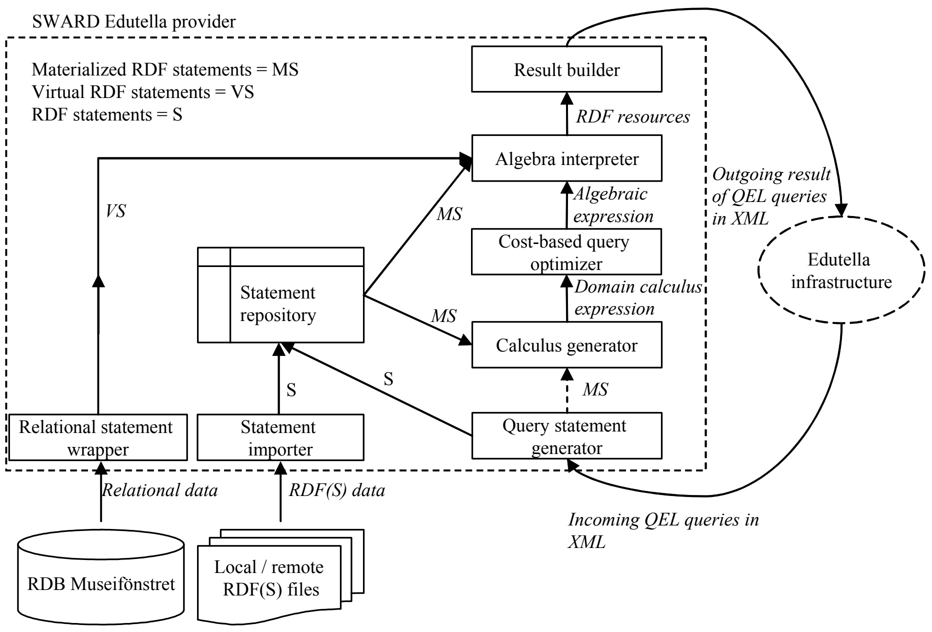
qel:s(x,p,t) is true if there is an RDF statement matching the triple <x,p,t> where x, p, and t are variables bound to resources. t may also be a literal. qel:like(t,'Matter') is true if either t is a literal and the string value of t contains the string 'Matter', or t is a resource and the URI of t contains the string 'Matter'.

q1 is intentionally chosen to be very simple to enable, for the reader, a perceivable step-by-step translation of the query to SQL in later sections. However, in an experiment measuring processing time of a QEL query in SWARD, described in section 6, a more complicated query containing a join is used.

4 System Architecture

Fig. 1 describes the architecture of SWARD. A QEL query arrives at the SWARD Edutella provider. There is a *query statement generator* building on Jena2 that parses incoming RDF data serialized as XML and extracts existing RDF statements expressing QEL queries. The dotted arrow in Fig. 1 from the query statement generator to the *calculus generator* indicates logical flow of execution. Actually, the parsed RDF statements are first stored in the local *statement repository*. The calculus generator then translates the materialized RDF statements corresponding to a specific QEL query into a *domain calculus expression*. It includes a fix-point rewrite algorithm to minimize the domain calculus expression, before it is translated by the *cost-based query optimizer* into an algebraic expression. This algebraic expression is then interpreted by the algebra interpreter over the combination of *materialized* RDF(S) statements, explicitly imported and stored by the *statement importer* in the statement repository, and *virtual* RDF statements generated by the *relational statement wrapper*.

[2] @prefix notation used to abbreviate namespaces.



SWARD Edutella provider
Materialized RDF statements = MS
Virtual RDF statements = VS
RDF statements = S
Result builder
RDF resources
Algebra interpreter
Algebraic expression
Cost-based query optimizer
Domain calculus expression
Calculus generator
MS
Query statement generator
VS
MS
MS
Statement repository
S
S
Relational statement wrapper
Statement importer
Relational data
RDF(S) data
RDB Museifönstret
Local / remote RDF(S) files
Outgoing result of QEL queries in XML
Edutella infrastructure
Incoming QEL queries in XML

Fig. 1. Architecture of SWARD

Thus materialized statements form a local database in SWARD while virtual statements are views of data in external sources. Instances of virtual statements are dynamically created and streamed through the system. A garbage collector automatically removes no longer needed virtual statement instances. Finally, the result of executing the algebraic expression is sent back over Edutella as variable-resource or variable-literal bindings serialized as XML by the *result builder*.

Examples of materialized statements are statements imported from files containing RDF(S) data such as the W3C definition of RDF(S) as meta-data statements. Hence, as illustrated in Fig. 1, data in SWARD statement repository can originate from local or remote RDF(S) files or the Edutella infrastructure.

Fig. 2 illustrates the modeling of wrapper data sources in SWARD. SWARD extends the basic RDFS model with RDFS classes representing different *statement sources*[3], and the possibility to define a hierarchy of such sources.

An RDFS class acting as a statement source is instantiated only once by the system upon initialization. Each statement source has an associated property, `stmts`, maintained by the system that generates the statement set of RDF statements in the source. Notice that statement sets belong to a source while the resources referenced by the statements are universal.

[3] A statement source is a data source with its content translated into RDF statements. Observe that a RDFS class representing a statement source is modeling a data source and not the semantics of data in that data source.

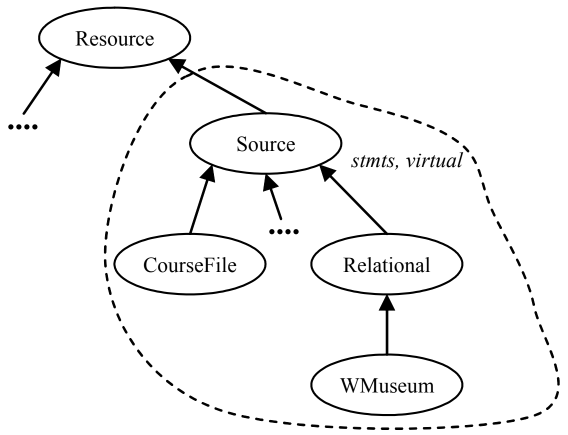


Fig. 2. Hierarchy of wrapper data sources in SWARD

There is a hierarchy of statement sources to handle that some sources are specializations of other, e.g. `WMuseum` is a specialization of general relational database statement sources. Statements in a statement source, *s*, are seen as a union of statements in statement sources subclassing *s*. The root class `Source` represents all RDF statements. Each other RDF statement source is a subclass of `Source`. There is a subclass to `Source` called `Relational` representing all relational database statement sources. As illustrated by statement source `CourseFile` we also allow other kinds of sources than relational databases. `CourseFile` represents RDF files containing courses read by computer science students at Uppsala University. This paper focuses on relational data sources which are subclasses to class `Relational`. For example, class `WMuseum` represents the specific relational statement source *WM*. This separation between `Relational` and its subclasses enables us to generate and compute tailor made statement sets for different databases. For example, often, for scalability reasons, SWARD should treat statements in a statement source representing a relational database as virtual statements. Therefore RDFS classes representing statement sources have a Boolean valued property `virtual` indicating if the statements belonging to a source are virtual or not. For the RDFS class to be virtual all its subclasses have to be virtual. The statement hierarchy can easily be extended with additional statement sources.

5 RDF Views over Relational Data

As illustrated in Fig. 1 a QEL query is translated from XML to an intermediate domain calculus representation. For each table $T(C_1,\dots C_n)$ in relational database R where column named C_1 is key for simplicity, SWARD generates a set of views denoted $CSS(R, T, C_i)$ with definitions:

$$\text{CSS}(R,T,C_i): \{s,p,o | s=\text{uriKey}(R,T,c_1) \wedge p=\text{uriCol}(R,T,C_i) \wedge T(c_1,\ldots,c_n) \wedge o=c_i\} . \quad \textbf{(1)}$$

where *uriKey(R, T, c_1)* computes a unique URI for the key c_1 and *uriCol(R, T, C_i)* denotes a unique URI for the column named C_i. Notice that *CSS(R,T,C_i)* describes a column C_i, i.e. it is not materialized. The name of the view is generated by concatenation, e.g. `CSS('WM','Resource','Name') = WMResourceName`. Table I shows how URIs representing data from a relational database are auto-generated by the system, given a user defined namespace, *ns*[4].

Table 1. Schema for autogenerating URIs representing data from a table *T(C_1,...,C_n)* in a relational database *R* using namespace *ns*

Function	**Generated URI**
uriCol	*ns:databasename.relationname.columnname*
uriKey	*ns:databasename.relationname.columnname keyvalue*
compUriKey	*ns:databasename.relationname. columnname$_1$...columnname$_n$. keyvalue$_1$...keyvalue$_n$*

There are cases when URIs should be user specified rather than automatically generated. For this SWARD allows the user to explicitly specify *uriKey* for a CSS. For example, the table `Resource` in *WM* already includes a field, `URI`, containing unique URIs for each row (a secondary key) and this column is therefore chosen by the user to represent the *uriKey* in the definition of `WMResourceName` as illustrated in Example 1. Notice that the term `Resource` represents the wrapped relational table:

```
WMResourceName(s,p,o):

{s,p,o|
s=uri∧                          /*uriKey*/
p='ns:wm.resource.name'∧        /*uriCol*/
Resource(rid,mid,name,uri,shortdesc,desc)∧
o=name}
```

Example 1. Definition of WMResourceName

The algebra generator will combine CSSs appearing in a QEL query and generate SQL strings for accessing the wrapped database. Values from column named C_i are treated as literals if the column is not a foreign key. If C_i were a foreign key from another table in *R*, *T'*, the system would replace '`o=name`' in Example 1 with '`o=uriKey(R,T',name)`'. (Compound keys are treated as tuples and handled by the function *compUriKey*).

[4] Namespace:'http://www.museifonstret.se/'

The statement set, `stmts`, of a statement source that represents a relational database is defined as the union of all column statement sets for the source.

Fig. 3 illustrates how the statement set of `WMResourceName` is defined from column `Name` in table `Resource` in *WM*.

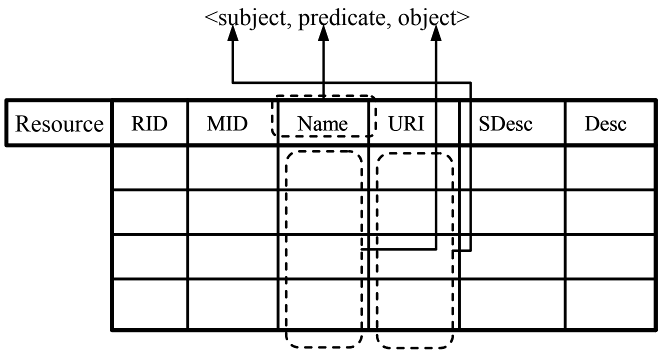


Fig. 3. Producing the statement set of `WMResourceName`

SWARD allows for different terminologies in received QEL queries and in URIs from the CSSs. In our example Edutella uses a terminology based on Dublin Core, which is different from the terminology of statement source `WMuseum`. The relations between two URIs from different terminologies having the same meaning are called *source mappings*. They are represented by a user defined table `SM(U1, U2)` in the wrapper taking two URIs `U1` and `U2` as arguments and evaluating to true if there is a source mapping between them.

For example, the RDF predicate produced by `WMResourceName` in Example 1 is mapped to the URI '`dc:title`' in Dublin Core.

6 Translation of QEL to Optimized SQL

QEL queries are specified against an RDF view containing both materialized statements from SWARD statement repository and virtual statements mapped from the wrapped relational database. In our example, *q1* is represented by the following domain calculus expression:

$$\{\texttt{s,p,o}\,|\,\texttt{stmt(s,p,o)}\wedge\texttt{like(o,'Matter')}\wedge\texttt{p='dc:title'}\}$$

`stmt` and `like` implements the built-in QEL predicates `qel:s` and `qel:like`, respectively. `stmt` is evaluated over the statement set *w* of statement source `Source`. `stmt(s,p,o)` evaluates to true if there is an RDF statement <s, p, o> in `stmts` of `Source`.

In the rest of this section, for simplicity, *w* is assumed to be equal only to RDF statements in `WMuseum`. Furthermore, *WM* is restricted to contain only one table, Resource. Hence *w* can be seen as the disjunction of all CSSs in WMuseum and *q1* is expanded to the following expression:

```
{s,p,o|
(WMResourceRID(s,q,o)∧SM(q,p))∨
(WMResourceMID(s,q,o)∧SM(q,p))∨
(WMResourceName(s,q,o)∧SM(q,p))∨
(WMResourceURI(s,q,o)∧SM(q,p))∨
(WMResourceShortDesc(s,q,o)∧SM(q,p))∨
(WMResourceDesc(s,q,o)∧SM(q,p))∧
p='dc:title'∧like(o,'Matter')}
```

The resulting calculus expression is transformed into a simpler one by a fix point algorithm using rewrite rules [9]. Thus the above expression is first translated to *disjunctive normal form*:

```
{s,'dc:title',o|
(WMResourceRID(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceMID(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceName(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceURI(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceShortDesc(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))∨
(WMResourceDesc(s,q,o)∧SM(q,'dc:title')∧
like(o,'Matter'))}
```

Example 2. Expression of *q1* on disjunctive normal form

Each CSS is substituted for its definition and simplified. For the CSS WMResource-Name, according to Example 1, this produces the following term in the disjunction above:

```
Resource(rid,mid,o,s,shortdesc,desc)∧
SM('ns:wm.resource.name','dc:title')∧
like(o,'Matter')
```

Example 3. Substitution of WMResourceName for its definition

With *compile time evaluation* the SM table is then evaluated by the *calculus generator* and replaced with TRUE since SM maps 'dc:title' to 'ns:wm.resource.name'. For the other terms in the disjunction (shown in Example 2) SM will be evaluated to FALSE and they will be removed. The only remaining calculus term is then translated into an algebraic expression, or query plan, by the *cost-based query optimizer* that contains calls to a foreign function executing SQL according to Fig. 4. This shows that compile time evaluation substantially reduces the size of the calculus expression that is sent to the *cost-based query optimizer* and therefore reduces the query optimization time.

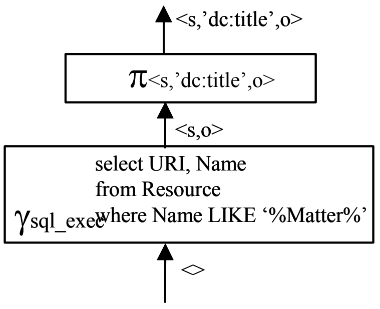


Fig. 4. An execution plan for the QEL example query

An algebra operator is one of {π, σ, ×, ∩, ∪, ⋈, γ}[9]. The π, σ, ×, ∩, ∪, and ⋈, operators have the same semantics as their relational counterparts. The γ (generate) operator performs function application. It can thereby introduce objects other than those produced by the leaf nodes into the query plan. In this way, the γ operator is similar to the generate operator in [24]. The function `sql_exec` sends an SQL query to a relational database.

An essential technique for improving the efficiency is to push down filter operators such as `like` to SQL when possible as in Fig. 4. The system has special rewrite rules for generating SQL strings from the calculus and it knows what functions can be executed in the sources, e.g. `like`, and generates SQL strings with calls to such functions. Furthermore, SWARD uses the heuristics to generate from a term as few SQL queries as possible but never a Cartesian product or a union.

In *q1* the variable `p` in the built-in predicate `qel:s(x,p,t)` is known to be equal to '`dc:title`'. This enables SWARD to drastically reduce the number of clauses in the disjunction shown in Example 2 using compile time evaluation of the `SM` table. Once `SM` is evaluated there is only a single clause left (see Example 3) from the original disjunction which is translated to a single SQL expression. An interesting situation arises when `p` is unknown. In SQL the column names of a query must be explicitly specified. This is critical for relational database query optimization. By contrast RDF queries can have dynamic properties, i.e. RDF queries can contain variables bound to RDF predicates. This would correspond to variables bound to table columns in SQL, which is not allowed. For QEL this means that the calculus predicates such as `qel:s` may be constructed out of variables rather than of known RDF resources.

A *statement cache* is used in SWARD to recycle already compiled QEL queries meaning that the query can be executed directly without relational query optimization.

Table 2. Measuring execution time of *q2* over variable sized table `Resource` without compile time evaluation

Tuples	**Proc**	s_{Proc}	**Exec**	s_{Exex}
700	8.667	0.013	0.005	0.008
2000	8.802	0.015	0.010	0.009
5000	8.786	0.027	0.020	0.009
10000	8.776	0.023	0.016	0.007

To see how compile time evaluation can improve performance an experiment was conducted on a PC with a Pentium 3 2.2 GHz processor with 512 RAM running *Microsoft SQL server*. A QEL query was executed repeatedly over the relational database *WM* with one table, `Resource`. The table was scaled up in each test with 700, 2000, 5000, and 10000 tuples. For every table size the test was made first without, see Table II, and then with compile time evaluation, see Table III. The query used in the experiment, *q2*, is an extension of the query in our running example and is expressed in QEL as:

```
@prefix qel:http://www.edutella.org/qel#
@prefix dc:http://purl.org/dc/elements/1.1/
?(x,t,d):-qel:s(x,'dc:title',t),
          qel:s(x,'dc:description',d),
          qel:like(t,'Matter'),
          qel:like(d,'Fysik')
```

In natural language that would be: Find all museum artifacts that have a name that contains the string '`Matter`' and a description that contains the string '`Fysik`'[5].

The time for each test was measured as the time taken to process the query; **Proc** and the time taken to execute the query; **Exec** as part of **Proc**. For the purpose of our experiment and for simplicity, **Exec** was defined as the time spent calling SQL. All tests were measured in seconds. Table II and Table III respectively show the results of the experiment with or without compile time evaluation. For each test the mean value was calculated and chosen as the result of the test. Standard deviations for **Proc** (s_{Proc}) and **Exec** (s_{Exec}) were also calculated.

Table 3. Measuring execution time of *q2* over variable sized table `Resource` with compile time evaluation

Tuples	**Proc**	s_{Proc}	**Exec**	s_{Exec}
700	0.953	0.017	0.010	0.009
2000	1.130	0.032	0.020	0.005
5000	1.140	0.027	0.020	0.006
10000	1.115	0.025	0.020	0.006

When analyzing the result of the experiment we see that there is a noticeable reduction in time spent processing *q2*, **Proc**, when using compile time evaluation

[5] English:'Physics'.

compared to when not. However, the execution time, **Exec**, is more or less constant. The experiment shows that query processing time is improved substantially by compile time evaluation of the SM table. However, the query execution time **Exec** is not affected by compile time evaluation. The reason is that the query optimizer in SWARD knows that `SM` is local to the wrapper. Therefore it is evaluated before accessing the more expensive relational back-end and this makes query execution efficient.

7 Summary

The SWARD system provides RDF views over relational databases in terms of *virtual* statements. Transformations according to some general translation rules yield optimized RDF queries in terms of domain calculus expressions. An algebra generator then produces a query plan out of these expressions that contains calls to functions executing SQL queries.

Various techniques are used to optimize the expressions such as rewrites and push down of filter operators (e.g. `like`) to the relational database. This allows for the creation of RDF views over relational databases in a highly scalable way.

When translating semantic web queries to SQL there is a problem that, unlike SQL, semantic web queries are dynamic and they are not necessarily expressed in terms of a schema. However, SWARDs abstraction of relational database columns into CSSs makes the translation of QEL queries into SQL natural. To execute a QEL query, q, over an RDF view w means evaluating $w \wedge q$ as illustrated in Example 2. To retrieve the entire view means executing all these expressions and appending their results. Thus RDF predicates become large disjunctions of clauses where each clause is a conjunction of constraints in the QEL query and calls to the relational database. Since the disjunctions are large it is important to reduce their size before generating the query algebra expression. We have shown that compile time evaluation of the source mappings between URIs of different terminologies allows for substantial reduction of the calculus expression and improved query compilation time. However, in our example the query execution time is not effected since the regular cost-based query optimization produces an efficient execution plan in any case.

We are currently generalizing our approach to include more complex queries and other optimization strategies.

SWARD provides views over relational databases only in terms of basic RDF data. Future work will offer a semantically enriched RDFS form. This requires creating for each statement source some additional virtual statements providing information about class-subclass relationship, instance-of etc.

References

1. Barrett et al.: RDF Representation of Metadata for Semantic Integration of Corporate Information Resources, *Proc. WWW2002*, 2002.

2. D.Beckett and J.Grant: SWAD-Europe: Mapping Semantic Web Data with RDBMSes, *http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/*, 2001.
3. T.Berners-Lee, J.Hendler, and O.Lassila: The Semantic Web, *Scientific American*, May 2001.
4. D.Brickley and R.V.Guha: RDF Vocabulary Description Language 1.0: RDF-Schema, *http://www.w3.org/TR/2004/REC-rdf-schema-20040210/*, 2004.
5. C.Bizer: D2R MAP - A Database to RDF Mapping Language, *The 12th International World Wide Web Conference (WWW2003)*, Budapest, Hungary, 2003.
6. V.Christophides, G.Karvounarakis, A.Magkanaraki, D.Plexousakis, and V.Tannen: The ICS-FORTH Semantic Web Integration Middleware (SWIM), *Data Engineering Bulletine*, IEEE, 26(4), Dec. 2003.
7. S.Decker et al.: The Semantic Web - on the Roles of XML and RDF, *IEEE Internet Computing*, Sept./Oct. 2000.
8. Dublin Core Meta-data Initiative, Dublin Core Metadata Element Set, V 1.1, *http://dublincore.org/documents/dces/*
9. G. Fahl and T. Risch: Query Processing over Object Views of Relational Data, *The VLDB Journal*, Springer, Vol. 6, No. 4, 261-281, 1997.
10. H. Garcia-Molina et al.: The TSIMMIS Approach to Mediation: Data Models and Languages, *Intelligent Information Systems (JIIS)*, Kluwer, 8(2), 117-132, 1997.
11. L. Haas, D. Kossmann, E.L. Wimmers, and J. Yang: Optimizing Queries across Diverse Data Sources, *Proc. 23rd Intl. Conf. on Very Large Databases (VLDB'97)*, 276-285, 1997.
12. V. Josifovski and T. Risch: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations, *Proc. 25th Conference on Very Large Databases (VLDB'99)*, 435-446, 1999.
13. G.Karvounarakis el al.: Querying the Semantic Web with RQL, *Computer Networks and ISDN Systems Journal*, 42(5), 617-640, August 2003.
14. G.Klyne and J.J.Carroll: Resource Description Framework (RDF): Concepts and Abstract Syntax, *http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*, 2003.
15. L.Liu and C.Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources, *Distributed and Parallel Databases*, Kluwer, 5(2), 167-205, 1997.
16. A.Magkanaraki, V.Tannen, V.Christophides, and D.Plexousakis: Viewing the Semantic Web Through RVL Lenses, *2nd International Semantic Web Conference (ISWC'03)*, Sanibel Island, Florida, USA, 2003.
17. *http://www.museifonstret.se/*
18. W.Neidl et al.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. *Proc. 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002.
19. D. Quass, A. Rajaraman, Y. Sagiv, J.Ullman, and J. Widom: Querying Semistructured Heterogeneous Information in Deductive and Object-Oriented Databases, *Proc. of the DOOD'95 conference*, LNCS Vol. 1013, 319-344, Springer 1995.
20. RDF Gateway - a platform for the semantic web, Intellidimension, *http://www.intellidimension.com/*.
21. RDF Query Exchange Language (QEL) - concepts, semantics and RDF syntax, *http://edutella.jxta.org/spec/qel.html*
22. 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 Computing with Data*, Springer, 2003.
23. A.Seaborne: RDQL - A Query Language for RDF, W3C Member Submission, *http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/*, 2004.
24. DD.Straube, and MT.Özsu: Queries and query processing in object oriented database systems, *ACM Transaction Information Syst*, 8(4), 1990.

25. A. Tomasic, L. Raschid, and P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO, *IEEE Transactions on Knowledge and Date Engineering*, 10(5), 808-823, 1998.
26. K.Wilkinson, C.Sayers, H.A.Kuno, and D.Reynolds: Efficient RDF storage and retrieval in Jena 2, *1st Intl. Workshop on Semantic Web and Databases (SWDB'03)*, Berlin, http://hplabs.hp.com/techreports/2003/HPL-2003-266.html, 2003.
27. J. Ullman: *Database and Knowledge Base Systems*, Vols 1 & 2, Computer Press, 1988.