

Name: Andrew Newman

Student Number: 41227462

Supervisor: Jane Hunter

Project Title: Querying the Semantic Web using a
Relational Based SPARQL

Table Contents

INTRODUCTION	3
LITERATURE REVIEW	4
METHODOLOGY	6
PRELIMINARY STUDY USING A SIMPLE EXAMPLE	7
PLAN AND TIMELINE FOR PROJECT	10
Semester 1	11
Semester 2	11
REFERENCES	11

Tables and Figures

FIGURE 1. THE JRDF REPRESENTATION OF RDF NODES AND TYPES. THE NODE VALUE TYPES: URI REFERENCE (URI), BLANK NODE (BNODE) AND LITERAL EXTEND THE POSITIONAL TYPES: SUBJECT, PREDICATE AND OBJECT.....	8
TABLE 1. RDF AND RELATIONAL COMPONENTS	8

Introduction

The Semantic Web is an attempt to allow data from different sources to be combined in a consistent way by extending the current World Wide Web. It is useful when the data schemas are heterogeneous, change over time, or consensus on a common format cannot be reached ahead of time. It is used in areas such as Bio-Informatics, Life Sciences, GIS (Geographic Information Systems), Knowledge Management and Material Sciences.

A part of the W3C's Semantic Web initiative is RDF (Resource Description Framework). RDF is a simple graph-based data model for representing information on the Web [1]. It has a formal data model with formal semantics and is designed to be simple, open, and extensible [1]. An RDF graph consists of a set of triples; each triple consists of a Subject, Predicate and Object. Triples are used to create relationships between the subject and object using different predicates [1].

SPARQL (SPARQL Protocol And Query Language) is the W3C's proposed standard for querying RDF [2]. Similar to other query languages SPARQL allows users' to declaratively specify the conditions required for data to be retrieved rather than explicitly describing the individual steps required to return the data. SPARQL provides:

- Simple matching of RDF data,
- The ability to combine multiple matches together,
- Matching data types such as integers, literals, etc. based on conditions such as greater than, equal to, etc.
- Optionally matching data – that is, if certain data does exist it must meet a certain criteria but does not fail if the data doesn't exist,
- Combining RDF data sets together to query at the same time, and
- Ordering and limiting matched data.

A formal set-based model for SPARQL has not currently been provided. The lack of a formal set-based model means that there exist some incompatibilities between SPARQL and the RDF model.

To provide a compatible set-based model an existing formal model, the relational model seems to be appropriate. This model has been used as the basis for database management and as defined by Date, it consists of three components: structure, integrity and manipulation [3].

An alternative is to use a common commercial implementation of a query language such as SQL (Structured Query Language). SQL has been reconstructed using bags (a collection of values that allow duplicates) rather than sets (a collection of values that allows no duplicates) [5]. The operations available in SQL such as DISTINCT and aggregate functions are only applicable as operations over bags not sets [5]. This is incompatible with the RDF model and for this reason SQL has not been used.

Previous work has centred on either mapping SQL to SPARQL [6][7], creating a formal algebra for RDF without relating it back to SPARQL [8], or providing an RDF query language without consideration of SPARQL [10][11]. There exist only incomplete examples of producing a formal model or a mapping to other formal models such as the relational model or SQL [6][7][8].

As both the RDF model and the relational model are set-based it is likely that a

compatible model for querying RDF can be provided. This will lead to two direct advantages for users and implementers:

1. It provides a formal model that unambiguously outlines a consistent set of principles to create a coherent foundation for the formulation of queries. This provides a stable set of fundamentals that remain constant as implementations or syntaxes evolve over time.
2. It allows the continuing work being done on the relational model to be applied to querying RDF.

An extensive review of the literature available on the subject indicates that when developing SPARQL implementations the use of the relational model is considered important as a way to leverage existing work. This is especially relevant concerning the efficient execution of queries [7][8].

However, it is also apparent that there are numerous difficulties in mapping between SPARQL operations and the relational model. The main issues are:

- The existence of SPARQL's DISTINCT operator,
- Differences between unbound and NULL values,
- Order dependency when processing constraints such as OPTIONAL, and
- Processing FILTER operations.

This project proposes to:

- Investigate extending the relational model to support RDF data,
- Specify a query language that closely follows the syntax and operations defined by the SPARQL specification while using an underlying relational model,
- Investigate improvements that can be made in performance, scalability, distribution and usability when querying Semantic Web data using the new query language,
- Compare the differences between the SPARQL specification and the relational model and create recommendations for changing the SPARQL specification to align it with the relational model, and
- Use an existing RDF API to add appropriate relational bindings and to map these operations to SPARQL syntax and operations.

This project is limited by time constraints and this may restrict the number of SPARQL features implemented in the working prototype, however a set of core features will be incorporated.

Literature Review

A data model defined by Date, "...provides an abstract, self-contained, logical definition of data structures, data operators and so forth, that together make up the abstract machine with which the user interacts" [4].

It is understood [10] that the "...underlying data model directly influences the set of operations that should be provided by a query language." And further it is highlighted that the design of an RDF query language should support:

- The RDF abstract data model,
- Formal semantics and inference,
- Support for XML schema data types and
- Support for incomplete or missing information.

The set of relational operators combined with the relational model are collectively called the relational algebra [3]. The operations on relations originally defined by Codd include: set operations, projection, join, Cartesian product, and restriction [4]. It is from these relational operations that you can derive a primitive set of restrictions that includes: restrict, project, join, union and semi difference [3]. This is reinforced by [12] where basic algebraic operations of selection, projection, Cartesian product, set difference and set union are highlighted. These operations are analogous with SPARQL operations.

While SQL does provide similar operations to relational algebra it has a number of problems [5], "...no one really knows what SQL is, since there are many different versions, it is widely accepted that any version of SQL has at least two features which are not present in the relational algebra: aggregate operators...[and] allows a limited form of nesting by using the GROUP-BY construct...one needs bag semantics for the correct evaluation of aggregate functions."

An alternative operation that is compatible with the set-base, relational model and used by Date [3] is the SUMMARIZE operation which provides summaries such as COUNT, SUM, MAX and MIN that "...are not aggregate operators, though most of them do have the same names as aggregate operators (SQL confuses the two notions, with unfortunate results)." SQL also allows duplicate values and can cause problems with both optimisation and query processing [3].

The main work to date dealing with mapping the relational model and SPARQL has been performed by Cyganik and Harris [7][8]. The motivation for their work is to make available the previous experience dealing with query planning and optimisation. Harris [8] specifically mentions that RDF implementations do not take advantage of database engines' built in query optimisers and knowledge of indexes.

Along similar lines, the paper by Frasincar et al [9], in rationalizing the requirement for an RDF algebra, states that issues of query optimisation "...are mostly neglected". The types of optimisations proposed include efficiently performing extraction of data and constructing results.

Cyganik [7] notes that NULLs in SPARQL lead to several inconsistencies with the relational model, "The SPARQL model does not, for example, distinguish between an OPTIONAL variable that is unbound in some solutions, and a variable that is not used in the query at all." This result leads to confusion as to what an unbound result means.

A parallel occurs in commercial databases using NULLs to indicate missing information [3]. The issues related to NULL values has been debated by Date and Codd [3], "...nulls have no place in the relational model. (Codd thought otherwise..." Henly [9] has applied this to the field of geoscience and notes that NULL information can be indicative of multiple reasons such as "not worthy of comment", "to be added later", "lost" or indicates that the value is between known ranges but has not registered significantly to be recorded.

SPARQL defines the OPTIONAL operator, as defined by Harris [7] "...is used to signify a subset of the query that should not cause the result to fail if it cannot be satisfied...it is roughly analogous to the left outer join of relational algebra." Again, Cyganik [6] has shown that the semantics of OPTIONAL is not compatible with relational algebra. SPARQL provides "only conflicts cause join failure" whereas

relational algebra provides "unbound variable causes join failure". This creates an order dependency on the use of OPTIONAL.

This is similar to the outer join feature implemented in Edutella, a peer-to-peer based system for performing distributed RDF searches [12]. Where, "In outer join all the answers terminate in true. You never get an empty answer. In cases where you ask questions with many variables and some do not match, you get an answer, but the result for those variables that did not match are filled by NULL." It was found that "...inconsistencies in the answers depend on which order the outer join body literals are interpreted in." It is clear that order dependency in executing queries leads to inefficient distribution and uncertainty in which results will be returned from a query.

A solution to efficiently and unambiguously implementing outer joins in relational algebra was presented by Galindo-Legaria [13]. It includes a "...hierarchical view of data...where relational attributes may be set-based" and "Instead of having one tuple with parent-child information for each child, we present the children as a set associated with the parent". This also effectively removes the requirement for NULLs as place fillers when tables of different attributes are joined.

Galindo-Legaria went on to show that outer join; or rather minimum union (otherwise know as "outer union") can be implemented using disjunctions. The advantages are that "disjunction is commutative and associative, which is a significant property for intuition, formalisms, and generation of execution plans..." [14].

Harris [6] further defines three cases where the relational model will have problems processing value constraints - specifically within FILTER operations. They are:

- Non-relational expressions such as regular expressions,
- Late bound expressions where variables are created outside their local blocks and
- Placing constraints in a required block on variables that are only bound in an OPTIONAL block.

Similarly, Cyganik [6] defines a problem for relational algebra where a selection, for example during a FILTER operation, can only access variables within it's own block.

Previous work has highlighted specific limitations of the current SPARQL implementations and the need for an underlying formal model. It has also been shown that the problems associated with current SPARQL operations can be solved using existing relational techniques. However, little work has been done in developing and evaluating an RDF query language that is built of formal set-based models while maintaining a focus on SPARQL. This is the focus of my project.

Methodology

This project intends to determine the feasibility of providing a formal model of SPARQL using the relational model. This includes:

- Demonstrating how SPARQL operations can be adapted to use the relational model,
- Extending the relational model using previously discovered methods,
- Developing a working prototype, and
- Comparing the working prototype with another SPARQL implementation.

SPARQL defines a series of operations, use cases and semantics. These will be modified to be consistent with an extended relational model and the operations of relational algebra. The mappings will then be contrasted with the existing SPARQL operations and any benefits associated will be highlighted.

The prototype query engine will provide a user interface to allow queries to be processed. The prototype will be developed using Java 5.0 and will include a user interface developed using Swing. This query engine will be implemented using the RDF library JRDF. This will include making the following modifications:

- Creating mappings from the RDF model to the relational model,
- Implementing the relational operations, and
- Performing SPARQL queries using the relational operations. For example, extending relational algebra to allow querying of missing data provided by the SPARQL operation OPTIONAL.

A comparative review, using a set of criteria to be defined, of the relational system and a non-relational system will be made. These differences are expected to include a more consistent model, improvements in performance, a greater ability to scale and distribute work, and an increased precision of the results returned. The sample data, based on FOAF or another data set, and a list of use cases based on the developed criteria, will be used to show the practical advantages of implementing a query engine in this way.

Preliminary Study using a Simple Example

An initial investigation into extending the relational model to store and query RDF has been performed. The purpose of which is to ascertain whether it is practical to use the relational model to make simple queries of RDF data.

In order to create a simple example, the components of the various systems must be clearly defined and contrasted. The various components include:

- The relational model - Consisting of Types, Attribute names, Attributes, Tuples, Headings and Relations [3].
- Relational Operators – A small selection including: Restrict, Project and Join.
- The RDF model - consisting of RDF triples that can further be broken down into positional elements: Subject, Predicate and Object [1]. Each of the positional elements can be of a certain type. Subjects are either *URI References* or *blank nodes*; predicates must be *URI References*; and objects can be *URI References*, *blank nodes* or *literals*.

Figure 1 shows the hierarchical structure of these node types as modelled by JRDF.

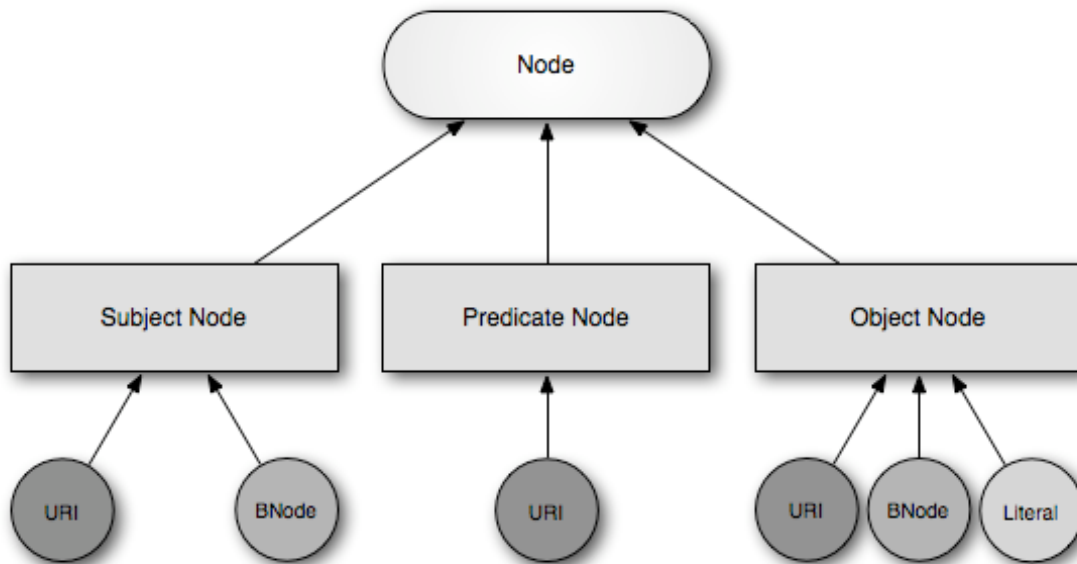


Figure 1. The JRDF Representation of RDF Nodes and Types. The node value types: URI Reference (URI), blank node (BNode) and literal extend the positional types: Subject, Predicate and Object.

Using this hierarchy of node types we can create a mapping between the relational and RDF components, as shown below in Table 1.

Component Name	Description	Relational Examples	RDF Examples
Type Name	A data type	integer, char, sno, name	subject, predicate, object, uri, literal and bnode
Attribute Name	A distinct, descriptive name	status, city, sno, sname	Variables: s?, ?city. Node Postions: subject, predicate, object
Attributes	A combination of type name and attribute name	status:integer, char:city, sno:sno, sname:name	s?:subject, p1:predicate, ?city:object
Tuple or Tuple Value	When a value is associated with an attribute	sno sno('s1'), sname name('smith'), status 20, city 'london'	s?:subject(#s1), p1:predicate(#name), o1:object('smith')
Heading	A set of attributes	sno sno, sname name, status integer, city char	s? subject, p1 predicate, o1 object

Table 1. RDF and Relational Components

A relation contains a heading and a body. The body contains a set of tuples. Using the typical supplier and part examples [3] the following relations have been created:

Suppliers

sno:sno	sp:sp
S1	P1
S1	P2
S2	P1
S2	P2

Parts

sp:sp	char:city
P1	'London'
P2	'Paris'

A similar representation using relations modified to use RDF components (RDF relations) is shown below:

Suppliers and Parts

s1:subject	p1:predicate	o1:object
S1	#sno	S1*
S1	#sp	P1
S1	#sp	P2
S2	#sno	S2*
S2	#sp	P1
S2	#sp	P2
P1	#sp	P1*
P2	#sp	P2*
P1	#city	'London'
P2	#city	'Paris'

* The current use of these S1 and P1 as Subjects may change in the future to be replaced by blank nodes.

A sample SPARQL query can be devised to return all the supplier numbers (sno), part numbers (pno) and cities.

Sample Query

```
select ?sno ?pno ?city
...
where ?sno #sno S1
?sno #sp ?pno
?pno #city ?city
```

This results in the following relations being created for each constraint (using relational restrict operation):

First Relation (?sno #sno S1)

?sno:subject	p1:predicate	o1:object
S1	#sno	S1

Second Relation (?sno #sp ?pno)

?sno:subject	p2:predicate	?pno:object
S1	#sp	P1
S1	#sp	P2
S2	#sp	P1
S2	#sp	P2
P1	#sp	P1
P2	#sp	P2

Third Relation (?pno #city ?city)

?pno:subject	P3:predicate	?city:object
P1	#city	'London'
P2	#city	'Paris'

By performing joins the following results are obtained:

Join of First and Second Relation

?sno:subject	P1:predicate	O1:object	P2:predicate	?pno:object
S1	#sno	S1	#sno	P1
S1	#sno	S1	#sno	P2

Join of First, Second and Third Relation

?sno:subject	P1:predicate	O1:object	P2:predicate	?pno:object	P3:predicate	?city:object
S1	#sno	S1	#sno	P1	#city	'London'
S1	#sno	S1	#sno	P2	#city	'Paris'

Using the relational project to produce only the variables defined in the SELECT part of the SPARQL query the final result is:

Query Result

?sno:subject	?pno:object	?city:object
S1	P1	'London'
S1	P2	'Paris'

This is consistent with the expected result when performing a SPARQL query using current SPARQL implementations.

Plan and Timeline for Project

The work to be completed has been divided by semester. The first set of tasks relates to creating the relational structures (tuples, relations, etc.) and providing a typing system to map the RDF model to the relational model.

The second set of tasks relates to creating the user interface in which queries will be submitted and allowing data to be loaded into the system. Furthermore, the complete set of relational operations will be mapped to SPARQL and any other provisions to be made to implement the queries.

This will then allow the comparison to be made using the test cases, test data and an existing non-relational implementation of SPARQL.

Semester 1

Time Frame	Tasks
Weeks 1 – 4	General project set-up – IDE, Wiki, build environment. Annotated bibliography.
Weeks 5 – 8	Initial investigation of mapping RDF to relational data types. Validation and review of design and creation of issue matrix. Literature review and survey of further required reading.
Weeks 9 – 13	Further development work on JRDF. Progress Report Draft and Delivery. Completion and review of SPARQL interpreter in JRDF. Begin initial implementation of relational operators. Draft Seminar and preparation. Presentation of Seminar. Finish demonstration of a subset of SPARQL operations mapped to the relational model. Constraint trees/diagrams describing how the relational operators perform SPARQL operations.

Semester 2

Time Frame	Tasks
Weeks 1-4	Completion of relational operators. Review of algebra for thesis. Finalisation of data sets to use as examples Optimisation and review of the code based on data sets and completed operators
Weeks 5-8	Initial investigation of non-relational system with selected data sets (FOAF or other data set). Begin work on poster and abstract Review and finalise UI work for demonstration purposes.
Weeks 9-13	Perform metrics and comparative evaluation of relational and non-relational systems. Complete and deliver poster and abstract. Complete and deliver thesis. Complete and deliver presentation.

References

- [1] G. Klyne and J. Carroll, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, World Wide Web Consortium (W3C) Recommendation, 10 February 2004; <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>
- [2] E. Prud'hommeaux and A. Seaborne, *SPARQL Query Language for RDF*, World Wide Web Consortium (W3C) Candidate Recommendation, 6 April 2006; <http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/>
- [3] C.J. Date, *Database in Depth, Relational Theory for Practitioners*, 1st ed., Sebastopol, California: O'Reilly Media, Inc., 2005, pp. 4–10, 41-57, 86-93.
- [4] E.F. Codd, "A Relational Model for Large Shared Databanks," *Commun. of the ACM*, vol. 13, no. 6, pp. 377–387, June 1970.

- [5] L. Libkin and L. Wong, "Query Languages for Bags and Aggregate Functions," *Journal of Computer and System Sciences*, vol. 55, no. 2, pp. 241-272, 1997; <http://citeseer.ist.psu.edu/libkin97query.html>
- [6] R. Cyganiak, "A Relational Algebra for SPARQL," Digital Media Systems Laboratory, HP Laboratories Bristol, Tech. Rep. HPL-2005-170, 2005; <http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html>
- [7] S. Harris and N. Shadbolt, "SPARQL Query Processing with Conventional Relational Database Systems," *WISE Workshops*, New York, NY, USA, pp. 235-244, Nov. 2005; <http://www.informatik.uni-trier.de/~ley/db/conf/wise/wise2005w.html>
- [8] F. Frasincar, et al, "RAL: An Algebra for Querying RDF," *World Wide Web, Internet and Web Information Systems (WWW)*, vol. 7, no. 1, pp. 83 - 109, 2004.
- [9] S. Henly, "The Man Who Wasn't There: Problems of Missing or Partially Missing Data in Geoscience Databases," *International Association for Mathematical Geology, IAMG 2003*, Portsmouth, UK, September 2003; http://www.iamg.org/meetings/Proceedings_2003/IAMG-2003.htm
- [10] P. Haase, et al, "A Comparison of RDF Query Languages," *Proc. of the Third International Semantic Web Conference*, Hiroshima, Japan, Nov. 2004; <http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf>
- [11] J. Bailey, et al, "Web and Semantic Web Query Languages: A Survey," *Reasoning Web, First International Summer School 2005*, Norbert Eisinger, Jan Maluszynski (editor(s)), LNCS 3564, 2005.
- [12] A. Pamukci, "Outer Join in Edutella," Master's thesis, Dept. of Numerical Analysis and Computer Science, Stockholm University, 2004.
- [13] C. Galindo-Legaria, "Algebraic Optimization of Outerjoin Queries," Doctoral diss., The Division of Applied Science, Harvard University, Cambridge, Massachusetts, June 1992.
- [14] C. Galindo-Legarai, "Outerjoins as Disjunctions," *Proc. of the 1994 ACM-SIGMOD Int. Conference on Management of Data*, Minneapolis, USA, pp. 348 - 358, 1994.