THE UNIVERSITY OF QUEENSLAND

School of Information Technology and Electrical Engineering

Querying the Semantic Web using a Relational Based SPARQL

by

Andrew Newman

The School of Information Technology and Electrical Engineering

The University of Queensland

Submitted for the degree of Bachelor of Information Technology (Honours)

25th October 2006

(Updated 7th November 2006)

Andrew Newman
Brisbane QLD Australia
25th October 2006

Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia QLD 4072

Dear Professor Bailes,

In accordance with the requirements of the Degree of Information of Technology (Honours) in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled:

"Querying the Semantic Web using a Relational Based SPARQL"

This thesis was performed under the supervision of Professor Jane Hunter. I declare that that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

____

(Andrew Newman)

Abstract

The Semantic Web is an initiative that aims to enable data from different sources to be combined in a consistent way. It is particularly useful when the schemas and terminologies of different data sets are to be merged; they differ between organisations or change over time. Semantic Web technologies have been successfully applied to data integration in fields such as Bio-Informatics, Life Sciences, GIS (Geographic Information Systems) and Material Sciences.

Resource Description Framework (RDF) is a simple graph-based data model for representing information on the Web. SPARQL is the proposed standard for querying RDF and both are part of the W3C's Semantic Web Activity.

The current SPARQL specification shows a strong bias towards underlying implementations such as SQL and lacks a formal model. Previous work by Cyganiak, Frasincar et al., Harris and Shadbolt, and Pérez et al. [5, 10, 16, 22] has highlighted the need for a formal model in order to improve consistency and clarity of the existing specification. A formal model would also allow independence between implementation and specification, greater consistency with RDF, and the ability to improve implementations without affecting the user's view of the system.

The relational model is a candidate for such a formal model as it is a mature formal model that is well understood and provides other features such as efficient query optimisation and distribution.

Building on previous work by Galindo-Legaria, this thesis provides a mapping from RDF and SPARQL using the relational model and shows the desirable outcomes of such an approach. The SPARQL operations UNION and OPTIONAL were implemented using the relational operators outer union and minimum union respectively. Implementation details using the JRDF library are described including the use of an order independent minimum union implementation. Some of the advantages in applying pre-existing relational optimisation techniques are explored.

It is shown that using the relational model as a basis for SPARQL provides an easier to implement, more efficient, more consistent and extensible query language than is currently provided. This approach allows the reuse of existing relational optimisation techniques and can be used as a basis to extend SPARQL functionality.

Acknowledgments

I would like to thank the following people for making this thesis possible:

- My family, especially Donna and Woz,
- My supervisor Jane Hunter,
- Brad Clow who supported the JRDF project on his company's time,
- Tom Adams who started SPARQL in JRDF and his work and feedback, and
- Paul Gearon, Simon Raboczi, Leo Sauermann, Andy Seaborne, and Robert Turner for their valuable feedback and thoughts.

CONTENTS

1 Overview

1.1 Introduction

Resource Description Framework (RDF) is a part of the W3C's Semantic Web initiative. RDF is a simple graph-based data model for representing information on the Web [18]. It has a formal data model, with formal semantics, and is designed to be simple, open, and extensible [18]. An RDF graph is a set of triples; each triple is made of a Subject, Predicate and Object. Triples are used to create relationships between the subject and object using different predicates [18].

SPARQL (SPARQL Protocol And Query Language) is the W3C's proposed standard for querying RDF [23]. SPARQL is one of a number of query languages designed to query formal representations of data such as XML (eXtensible Markup Language), Topic Maps and RDF which consist of data models represented as trees, topics and associations, and directed graphs respectively [1]. 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 definitions for:

- 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 the query 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.

The definitions listed above are somewhat loose, however. SPARQL has yet to be described formally using a set-based data model that follows RDF's abstract model. A data model, as defined by Date [6], "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".

Haase, et al. state that the "…underlying data model directly influences the set of operations that should be provided by a query language" [14]. Furthermore, it is highlighted that the design of an RDF query language should support [14]:

- The RDF abstract data model,
- Formal semantics and inference,
- XML schema data types and
- The ability to handle incomplete or contradictory information.

The beneficial properties of a query language for the Semantic Web defined by Bailey et al.

[1] include:

- Referentially transparent - “within the same scope, an expression always means the same”,
- Strong answer closure - the result of a query can be used as the input for further querying,
- Set-oriented functional – also known as a backtracking-free logic programming,
- Incomplete queries and answers - support for data on the Web that may not have defined schemas,
- Multiple serialisation aware - able to serialise data to various formats including XML, OWL, RDF and Topic Maps, and
- Queries that support reasoning capabilities - the ability to query different data sources and infer new statements.

The relational model is an existing model that could be used to provide a compatible set-based, formal model. This model has long been used as the basis for database management. Date defines it as consisting of three components: structure, integrity and manipulation [6]. It has been extended to support rules and inferencing [21], support for XML schema data types and other data types [8], to query hierarchical data [9] and to support merging data, potentially incomplete or contradictory information, through the use of outer joins and other techniques [13]. The relation model supports answer closure and referential transparency (for read-only queries).

The set of relational operators combined with the relational model are collectively called the relational algebra [6]. The operations on relations originally defined by Codd [4] include: set operations, projection, join, Cartesian product, and restriction. It is from these original operators that other relational operations have been derived including: restrict, project, join, and union [6]. In SPARQL they are analogous with: triples matches, SELECT, ‘.’ (join), and UNION.

An alternative to the relational model is SQL (Structured Query Language). SQL is often seen as an implementation of the relational model even though it has numerous incompatibilities with the relational model such as bag instead of set semantics, column ordering, duplicates and handling of nulls [6]. SQL has been formally reconstructed using bags (a collection of values that allow duplicates) rather than sets (a collection of values that allows no duplicates) [20]. From this work it’s shown that operations such as DISTINCT and aggregate functions are only applicable for bags and not sets. SQL’s use of duplicate values can also cause problems with both optimisation and query processing [6].

SQL also has other problems [20] such as,

> *“…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 it] allows a limited form of nesting by using the GROUP-BY construct…one needs bag semantics for the correct evaluation of aggregate functions.”*

Software that depends on SQL frequently has to adapt to each vendor specific implementation due to these differences.

Similarly, SQL's UNION operator has a number of problems in that it relies on a column ordering being used to match values rather than the columns being the same type (as defined by relational algebra) [7]. Date claims, "…given any two SQL tables, there are typically many distinct tables that can all be regarded as a union between two given SQL tables".

The distinct differences between SQL and RDF are the reason why SQL is not a natural choice as the basis of an RDF query language. It is clear than any formal SPARQL definition should abstract away any dependence from SQL and be solely based on the data model it is querying, RDF.

SQL does have a large industry following so it is crucial that a mapping from SPARQL to SQL exists. Work on this mapping has already occurred [2], but further work, especially using known SQL optimisation techniques [19], has yet to occur.

Previous work has highlighted specific limitations of the current SPARQL specification [5, 10, 16, 22] and subsequent implementations. To overcome these issues an underlying formal model should be established. However, little work has been done in developing and evaluating an RDF query language that is built on formal set-based models while maintaining a focus on SPARQL.

As both the RDF model and the relational model are both propositional and set-based it is likely that a compatible model for querying RDF can be provided. This should 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.

2 Related Work

2.1 Introduction

The main work to date dealing with mapping SPARQL to the relational model has been motivated, in part, to efficiently answer queries. The use of NULL, duplicates and the order in which operations are executed have been the main incompatibilities highlighted so far. With a clear definition of SPARQL operations such as UNION and OPTIONAL, previous work on efficient query implementations is highlighted.

2.2 Motivations for Creating a Formal Model

SPARQL has previously been compared and mapped to the relational model by Cyganiak [5], and Harris and Shadbolt [16]. A shared motivation for these works has been to make available the previous experience dealing with query planning and optimisation. Harris specifically mentions that RDF implementations do not take advantage of database engines' built in query optimisers and knowledge of indexes [16].

Pérez et al. [22] developed a formal model in an attempt to remove any redundancy or contradictions within the SPARQL specification. They also shared a similar motivation with previous relational mapping work, to reuse existing work on query planning and optimisation.

Along similar lines, the paper by Frasincar et al. [10] when rationalizing the requirement for an RDF algebra stated that issues of query optimisation "are mostly neglected". The types of optimisation proposed by Frasincar et al. included efficiently performing extraction of data and constructing results [10].

2.3 The Use of NULL

Cyganiak and Pérez et al. both note that unbound (or NULL values) in SPARQL lead to several inconsistencies with the relational model. Cyganiak says, "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 SQL databases through the use of NULLs to indicate missing information [6]. The issue related to NULL values has been debated by Date and Codd [6], "nulls have no place in the relational model." NULLs can have a direct impact on practical applications. For example, Henly [17] has applied criticism of NULLs to the field of geoscience and notes that NULL information can be caused by 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.

2.4 Duplicates

The existence and semantics of SPARQL operations such as DISTINCT and UNION appear to be based on underlying implementations, especially SQL, rather than following RDF semantics. RDF semantics clearly define sets of statements that express a proposition [18]. SPARQL, on the other hand, describes the possibility of duplicates and it makes no guarantee of a consistent approach to returning duplicates [23]. The same duplicate proposition is a tautology, “If something is true, saying it twice doesn’t make it any more true.” [6].

Date’s main prescription against duplicates [6] is: “…certain expression transformations, and hence certain optimizations, that are valid in a relational context are not valid in the presence of duplicates.” He presents [6] twelve different queries that produce nine different results each with their own degree of duplication. He concludes that, “you should always ensure that query results contain no duplicates…by always specifying DISTINCT in your SQL queries…if you follow this advice…there’s no good reason for allowing duplicates”.

2.5 Compositional vs Operational Semantics

A lot of work has been done since the inception of the relational model [24]. The foundation of this work has been accomplished by constructing operator trees to define a search space. The search space is explored for the most efficient operations by replacing operations with equivalent ones or by changing the order of execution. This is generally possible because of mathematical properties of the relational operators such as commutativity and associativity. In traditional optimisers such as System-R and Starburst a cost model of the query is created, operations are optimised and performed in a bottom-up fashion [24].

The work of Pérez et al. [22] highlights the distinction in SPARQL between two ways of executing queries: compositional and operational. Operational semantics consists of executing “a depth-first traversal of the parse tree of P and the use of the intermediate results to avoid some computations.” An example given is that the query (A OPT (B OPT C)) is executed by first executing A then B and then C. In other words, operational semantics executes a query in a left-to-right, top-to-bottom order. Compositional semantics executes the queries in a bottom-up order of execution. This would mean that the inner part of the example query, (B OPT C), is executed first.

Both Pérez and Cyganiak discuss the use of compositional over operational semantics when evaluating SPARQL queries. This choice of execution order is especially important for determining what the correct results of variables used both inside and outside of constraints that include an OPTIONAL clause (Cyganiak refers to it as “The “Nested OPTIONALs” Problem” [5]).

As relational implementations are based on compositional semantics, the rejection will reduce the ability to apply existing relational optimisation techniques. Processing query expressions to their conjunctive or disjunctive form for simplification and reordering is largely prevented as it depends on operations being associative and commutative.

2.6 The SPARQL OPTIONAL Operator

SPARQL's OPTIONAL operator as defined by Harris [16] "…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." Cyganiak [5] 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 relationship with left outer join indicates a similar order dependency for OPTIONAL.

The OPTIONAL functionality in SPARQL is similar to the outer join feature implemented in Edutella, a peer-to-peer based system for performing distributed RDF searches [21]. Edutella is based on Datalog that adds inference rules to relational databases,

> *"Outer join between R and S gives us all the tuples, whether they have matching tuples in the other relation or not. 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.

2.7 Optimisations Applicable to SPARQL

A solution to efficiently and unambiguously implementing outer joins in relational algebra was presented by Galindo-Legaria [12]. 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 has the advantage of removing the requirement for NULLs as place fillers when tables of different attributes are operated on. Instead, tuples contain sets of values for only those that are found or "…tuples in a relation may be defined on different sets of attributes, as long as they are a subset of the relation scheme".

The relational UNION operator was extended to be "untyped" (able to join on unequal relation schemes) called outer union [11]. The use of an outer union operation provides the similar semantics to SPARQL's UNION and is consistent with SPARQL's "only conflicts cause join failure" [5]. It also provides a formal, unambiguous definition of the operation.

Galindo-Legaria has shown that outer join can be implemented using minimum union to form join-disjunctive queries. The advantages are that "disjunction is commutative and associative, which is a significant property for intuition, formalisms, and generation of execution plans" [11].

The applicability of this using the approach suggested by Galindo-Legaria and Rosenthal was rejected by Pérez et al.:

> *"…classical results about outer join query reordering and optimization by Galindo-*

> *Legaria and Rosenthal are not directly applicable in the SPARQL context because they assume constraints on the relational queries that are rarely found in SPARQL…the assumption on predicates used for joining (outer joining) relations to be null-rejecting. In SPARQL, those predicates are implicit in the variables that the graph patterns share and by the definition of compatible mappings they are never null-rejecting…the queries are also enforced not to contain Cartesian products, situation that occurs often in SPARQL when joining graph patterns that do not share variables. Thus, specific techniques must be developed in the SPARQL context."*

It should be noted that the work of Galindo-Legaria forms the main basis of this thesis and contrasts with the claims made by Pérez et al. It is correct to state that simplification based on null rejection has not been used. This does not prevent optimising outer joins through tuple subsumption or using the definition of UNION proposed by Galindo-Legaria; specifically it allows unbound or NULL values within a tuple. Furthermore, it seems premature to discard the use of null rejection simplification in queries as it maybe possible to reuse these techniques by treating intermediate results using a slightly non-traditional approach. This approach is discussed in future work.

Another difference between Pérez et al. and Galindo-Legaria is the definition of left outer join. In Pérez et al. it is defined as:

$$\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$$

Where Ω is a set of mappings, where a mapping is a partial function of variables (V) to tuples (T). Tuples (T) consist of the positional elements subject (v1), predicate (v2) and object (v3) of an RDF statement.

Galindo-Legaria [12] defines left outer join of two relations R_1 and R_2 as:

$$R_1 ⟕ R_2 := (R_1 \bowtie R_2) \uplus (R_1 \rhd R_2)$$

Where, $\bowtie$ is the join of R_1 and R_2 on some predicate, $\uplus$ is outer union and $\rhd$ is antijoin. Antijoin is defined as:

$$R_1 \rhd R_2 := R_1 - (R_1 \ltimes R_2)$$

This is the difference between R_1 and the result of the semijoin ($\ltimes$) of R_1 and R_2. Where semijoin is:

$$R_1 \ltimes R_2 := \pi(R_1)\,(R_1 \bowtie R_2)$$

Where $\pi(R_1)$ is project on R_1 from the results of $\bowtie$ the join of R_1 and R_2.

The important distinction to make is that Pérez et al.'s definition uses the difference between two sets of tuples (\) rather than the antijoin. In Appendix A.5, Lemma 3 of [22] it is shown that:

$$(\Omega_1 \setminus \Omega_2) = \Omega_1 \setminus (\Omega_1 \bowtie \Omega_2)$$

The expanded version of left outer join becomes:

$$\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus (\Omega_1 \bowtie \Omega_2))$$

While Galindo-Legaria's left outer join with antijoin and semijoin expanded becomes:

$$R_1 ⟕ R_2 := (R_1 \bowtie R_2) \uplus (R_1 - (\pi(R_1)(R_1 \bowtie R_2)))$$

The only difference is the use of semijoin over join. While both of these operations produce the same overall result for left outer join, the use of semijoin is considered superior from the point of view of efficiency. This is highlighted by Galindo -Legaria [12]:

> ”*Since the join of relations R_1, R_2 applies some match predicate, it may not preserve all tuples from its arguments…This observations is the basis of some query processing algorithms to “reduce” relations…”*

Another simpler version of performing minimum union was also presented by Galindo-Legaria [11]. Instead of performing a series of operations that includes project, join, and antijoin, an approach using outer union followed by tuple subsumption called minimum union, was suggested. Minimum union is both commutative and associative and has a lower precedence than join. This is similar to the assertion in Pérez et al. that SPARQL's OPTIONAL has a lower precedence than its join operation [22].

Tuple subsumption is defined as t_1 subsumes t_2 if t_1 has more values that are not null than t_2 and that the values in t_2 that are not null are equal to t_1. The removal of subsumed tuples in R is denoted as $R\downarrow$. The minimum union of R_1 and R_2 is defined as:

$$R_1 \oplus R_2 := (R_1 \uplus R_2) \downarrow$$

Left outer join of R1 and R2 is then defined as:

$$R_1 ⟕ R_2 := R_1 \bowtie R_2 \oplus R_1$$

2.8 Other Issues with SPARQL

Harris [16] defines three other 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, Cyganiak [5] defines a problem for relational algebra where a selection, for example during a FILTER operation, can only access variables within it's own block.

3 Methodology

The feasibility of providing a formal model of SPARQL using the relational model will be explored by:

- 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 were adapted 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 and user interface was developed to allow queries to be processed. The prototype is written using Java 5.0 and includes a user interface developed in the Java Swing API. This query engine was implemented using the RDF library, JRDF. This will include making the following modifications:

- Mappings from the RDF model to the relational model,
- Creating a SPARQL parser,
- Relational constructs such as attributes, tuples and relations,
- Implementing the relational operations (Project, Restrict, Join, Union, Semidifference and Antijoin),
- Creating a Swing user interface,
- Adding the ability to execute relational operations in any order,
- Performing SPARQL queries using the relational operations including extending the relational algebra to allow querying of missing data provided by the SPARQL operation OPTIONAL, and
- Creation of tests to validate SPARQL compliance.

This considerably increased the complexity of JRDF with the number of Java classes required for implementation increased by nearly a factory of two from 155 to 292. There was an addition of approximately 15,000 lines of code from an existing base of 21,000 with an increase from 4857 to 9035 NCSS (Non Commenting Source Statements).

A comparative review is provided between the developed relational system using JRDF and Hewlett Packard's Jena library.

The sample data, based on FOAF data set and queries provided by the SPARQL test cases is used to show the practical advantages of implementing a query engine in this way.

4 Mapping RDF to the Relational Model

An initial investigation into extending the relational model to store and query RDF has been performed, with the purpose of ascertaining the practicality of using the relational model to make simple queries of RDF data.

In order to create a simple example of representing the RDF model in the relational model, the components of the various systems must be clearly defined and contrasted. These various components include:

- A concrete interpretation of a RDF graph - Each node in a graph, URI, Literal or bnode, is considered distinct if it has different values by character comparison or identity in the case of bnodes.
- The relational model - Consisting of Types, Attribute Names, Attributes, Tuples, Headings and Relations.
- 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. 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.

The following figure shows node value types: URI Reference (URI/IRI), blank node (BNode) and literal extend the positional types: Subject, Predicate and Object.

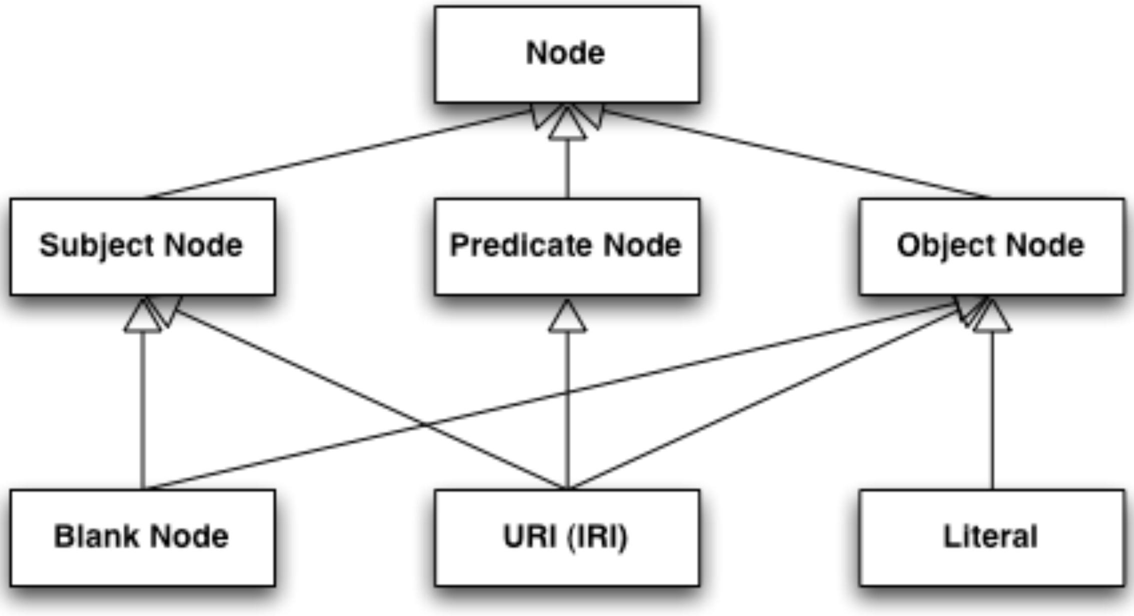

Figure 1. A UML Class Diagram of RDF Nodes and Types in JRDF.

Using this hierarchy of node types we can create a mapping between the relational and RDF components, as shown in the following table:

Component Name	Description	Relational	RDF/SPARQL
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
Attribute	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	A set of attribute and value pairs	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 and as defined by Galindo-Legaria, "tuples in a relation may be defined on different set of attributes, as long as they are a subset of the relation schema." [12].

4.1 Example

The following is a walk-through using the relational operations on RDF to produce the same results that are given by the SPARQL specification.

Using a limited set of sample tuples from the typical supplier and part examples given by Date [6] the following relations can be created:

Suppliers

sno:sno	sp:sp
Supplier1	Parts1
Supplier1	Parts2
Supplier2	Parts1
Supplier2	Parts2

Parts

sp:sp	char:city
Parts1	'London'
Parts2	'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
Supplier1	#sno	Supplier1*
Supplier1	#sp	Parts1
Supplier1	#sp	Parts2
Suppler2	#sno	Supplier2*
Suppler2	#sp	Parts1
Suppler2	#sp	Parts2
Parts1	#sp	Parts1*
Parts2	#sp	Parts2*
Parts1	#city	'London'
Parts2	#city	'Paris'

* Alternatively, Supplier1 and Parts1 as Subjects maybe modified to use blank nodes instead.

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 Supplier1 .
  ?sno #sp ?pno .
  ?pno #city ?city
}
```

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

Relation 1 (?sno #sno Suppler1)

?sno:subject	p1:predicate	o1:object
Supplier1	#sno	Supplier1

Relation 2 (?pno #city ?city)

?pno:subject	p3:predicate	?city:object
Part1	#city	‘London’
Part2	#city	‘Paris’

Relation 2 (?sno #sp ?pno)

?sno:subject	p2:predicate	?pno:object
Supplier1	#sp	Part1
Supplier1	#sp	Part2
Supplier2	#sp	Part1
Supplier2	#sp	Part2
Part1	#sp	Part1
Part2	#sp	Part2

By performing joins the following results are obtained:

Join Relation 1 and Relation 2

?sno:subject	p1:predicate	o1:object	p2:predicate	?pno:object
Supplier1	#sno	Supplier1	#sno	Part1
Supplier1	#sno	Supplier1	#sno	Part2

Join of Relation 1, 2 and 3

?sno:subject	p1:predicate	o1:object	p2:predicate	?pno:object	p3:predicate	?city:object
Supplier1	#sno	Supplier1	#sno	Part1	#city	‘London’
Supplier1	#sno	Supplier1	#sno	Part2	#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
Supplier1	Part1	‘London’
Supplier1	Part2	‘Paris’

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

4.2 RDF to Relational Types

JRDF’s query layer consists of a NodeType interface used to indicate the type of an attribute. It consists of a “composedOf” method that returns a set of node types. The node types supported consist of RDF positional types (SubjectNodeType, PredicateNodeType and ObjectNodeType), composite positional types (such as SubjectPredicateNodeType and all other unique combinations) and value types (BlankNodeType, LiteralNodeType and URIReferenceNodeType).

One of the first designs proposed to be used in JRDF was to calculate compositional nodes during joins. When two relations were joined and a variable appeared in two different places, a new node type was derived. For example, the join of two relations on a variable that appears both as a subject and a predicate results in it being associated with a new type "subjectpredicate". Where, "subjectpredicate" is a composite type made of a subject and predicate node type.

The current solution however, is to evaluate the query, record where the variables are being used in the query, and associate composite types to any variables located in more than one place. This process was adapted from an existing one where a check was being performed to ensure that all the variables in the SELECT clause are bound in the WHERE clause. This means that the join code does not have to infer new attribute types and the composite types are used up front when performing restrict operations.

4.3 Modifying Relational Joins

Two different types of joins, extensions of the original relational operations, were required for SPARQL support:

- Outer union - is a union that does not require both relations to be of the same matching headings or types.
- A null accepting join operation - relations of different types may be joined and a tuple is only rejected if bound variables have differing values.

The removal of matching types for union and a null accepting join may not seem to be initially compatible with the relational model. However, it is merely a convenience, and it is simply a combination of two or more relations. The following is an example of several minimum union operations as described on page 4 of Galindo-Legaria [11]:

CUSTOMERS	ORDERS	ITEMS'
C	O	I
C	O	-
C	-	-

This shows the result where the first row represents customers with orders for items, followed by customers with orders and then customers without orders. As described in the paper, projecting based on the required predicates can retrieve relations of the same type. Essentially, this is a representation of three traditional relations.

From this perspective, null accepting joins can be seen as an optimisation for performing operations on multiple relations in one pass.

The number of arguments, or arity, of operators in SPARQL is another example where the definition of operators is unclear. For example Pérez et al. [22] says, "using the binary operators UNION, AND and OPT, and FILTER", even though OPTIONAL is nadic (it has 1 or more parameters) in SPARQL's test cases [15]. In Tutorial D and the relational algebra defined by Date [6], join and union are both nadic. The associative nature of left outer join and hence the mapping of OPTIONAL to relational algebra (that does not allow order specified in its nadic operations) means there is a slight implementation difficulty. In JRDF, OPTIONAL operations are turned into a series of dyadic operations and if a monadic OPTIONAL operation is used it simply returns the initial binding.

Even though it is unclear if SPARQL's JOIN and OPTIONAL are truly nadic, JRDF supports nadic versions of these operations. In order to provide this support it requires the equivalent of TABLE_DEE (true) and TABLE_DUM (false) (the two 0 degree or nullary relations) to be defined. TABLE_DEE is the relation that contains one row, is analogous to true and is the identity with respect to JOIN. TABLE_DUM is the relation that contains no rows and is analogous to false. This also allows questions that result in a yes/no or true/false result to be returned.

The minimum union of TABLE_DEE and another relation is always just that relation. Using relational algebra the various combinations of the nullary relations (TABLE_DEE and TABLE_DUM or in JRDF RelationDEE and RelationDUM) in SPARQL operations are:

SPARQL Join

	DEE	DUM	Relation
DEE	DEE	DUM	Relation
DUM	DUM	DUM	DUM
Relation	Relation	DUM	Relation

SPARQL Union

	DEE	DUM	Relation
DEE	DEE	DEE	DEE
DUM	DEE	DUM	Relation
Relation	DEE	Relation	Relation

SPARQL Optional

	DEE	DUM	Relation
DEE	DEE	DEE	DEE
DUM	DUM	DUM	DUM
Relation	Relation	Relation	Relation

4.4 Order Independent Joins

An alternative version of OPTIONAL that was order independent but had similar semantics to OPTIONAL was investigated. As a suitable example, data was combined from FOAF files: http://clark.dallas.tx.us/kendall/foaf.rdf and http://eikeon.com/foaf.rdf. Sample queries were performed first following SPARQL semantics and then following the order independent semantics. For example, the query:

```
SELECT ?name ?mbox ?nick
WHERE {
  ?x <http://xmlns.com/foaf/0.1/name> ?name .
  OPTIONAL {
    ?x <http://xmlns.com/foaf/0.1/nick> ?nick
    OPTIONAL {
      ?x <http://xmlns.com/foaf/0.1/mbox> ?mbox
    }
  }
}
```

Using standard SPARQL semantics it produces:

?name:object	?mbox:object	?nick:object
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“k”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kclark”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kendallc”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“k”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kclark”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kendallc”
“Daniel Krech”		“eikon”

However, changing the order of the mbox and nick constraints produces a different result:

```
SELECT ?name ?mbox ?nick
WHERE {
  ?x <http://xmlns.com/foaf/0.1/name> ?name .
  OPTIONAL {
    ?x <http://xmlns.com/foaf/0.1/mbox> ?mbox
    OPTIONAL {
      ?x <http://xmlns.com/foaf/0.1/nick> ?nick
    }
  }
}
```

?name:object	?mbox:object	?nick:object
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“k”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kclark”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kendallc”

“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“k”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kclark”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kendallc”
“Daniel Krech”		

The important distinction to make is that the last row does not contain the nickname “eikon” - this is the expected behaviour.

If full outer join is used as the implementation behind the OPTIONAL operation then it is no longer order dependent. In turn, this provides the user with an easier way to express their requirements. The drawback to full outer join is that it produces results that contain a large number of more sparsely populated rows. Using the previous two queries the rows returned have mboxes and no name or just nicknames. Using the previous query the results are:

?name:object	?mbox:object	?nick:object
		“eikon”
“Daniel Krech”		“eikon”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“k”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kclark”
“Kendall Grant Clark”	mailto:kclark@ntlug.org	“kendallc”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“k”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kclark”
“Kendall Grant Clark”	mailto:kendall@monkeyfist.com	“kendallc”
	mailto:benkoo@ntr.net	
	mailto:bparsia@email.unc.edu	
	mailto:danbri@w3.org	
	mailto:edd@usefulinc.com	
	mailto:eikeon@eikeon.com	
	mailto:em@w3.org	
	mailto:gwachob@wachob.com	
	mailto:jim@jibbering.com	
	mailto:kendall@monkeyfist.com	
	mailto:khampton@totalcinema.com	
	mailto:libby.miller@bristol.ac.uk	
	mailto:me@aaronsw.com	
	mailto:niel@bornstein.atlanta.ga.us	
	mailto:uche.ogbuji@fourthought.com	

Note the rows with the values of “eikon” or “mailto:kendall@monkeyfist.com” which have the same bound values as other result but have more unbound values than other results returned. Subsumption can be used in order to remove these tuples.

The subsumption algorithm used in JRDF’s implementation was implemented by modifying the join algorithm. A tuple from the left hand side of the join was compared with each of the tuples on the right hand side. If any of them shared a common attribute/value combination further comparison was applied. In the case of the left hand side tuple having more values in it than the right hand side, and if the right hand side was a subset of the left hand side, then the right hand side tuple would be marked for removal.

4.5 Effect of Optimisation

A performance evaluation between two algorithms in JRDF and Jena’s in memory implementation was performed.

Jena and JRDF were run using Java 5.0 (version 1.5.0_06) under OS X 10.4.7 on a 2GHz Intel Core Duo with 2GB of main memory. Each query was performed 3 times and an average made. Before executing each query the Java process was restarted. In order to reduce the impact of garbage collection on the results a maximum and initial heap size was set to 1024MB.

The following queries were performed on a set of FOAF data consisting of 101 282, 50 822 and 25 520 statements.

Query 1

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?mbox ?name
{
  { ?x foaf:mbox ?mbox } UNION
  { ?x foaf:mbox ?mbox . ?x foaf:name ?name }
}
```

Query 2

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?mbox ?name
{
  ?x foaf:mbox ?mbox .
  OPTIONAL { ?x foaf:name ?name } .
}
```

Query 3

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?mbox ?name ?nick
{
  ?x foaf:mbox ?mbox .
  OPTIONAL { ?x foaf:name ?name } .
  OPTIONAL { ?x foaf:nick ?nick } .
}
```

The two algorithms in JRDF are implementations of OPTIONAL using either a combination of join, outer join and antijoin (labeled “JRDF” in the graph) or minimum union (labeled “JRDF w/ Sub.” in the graph).

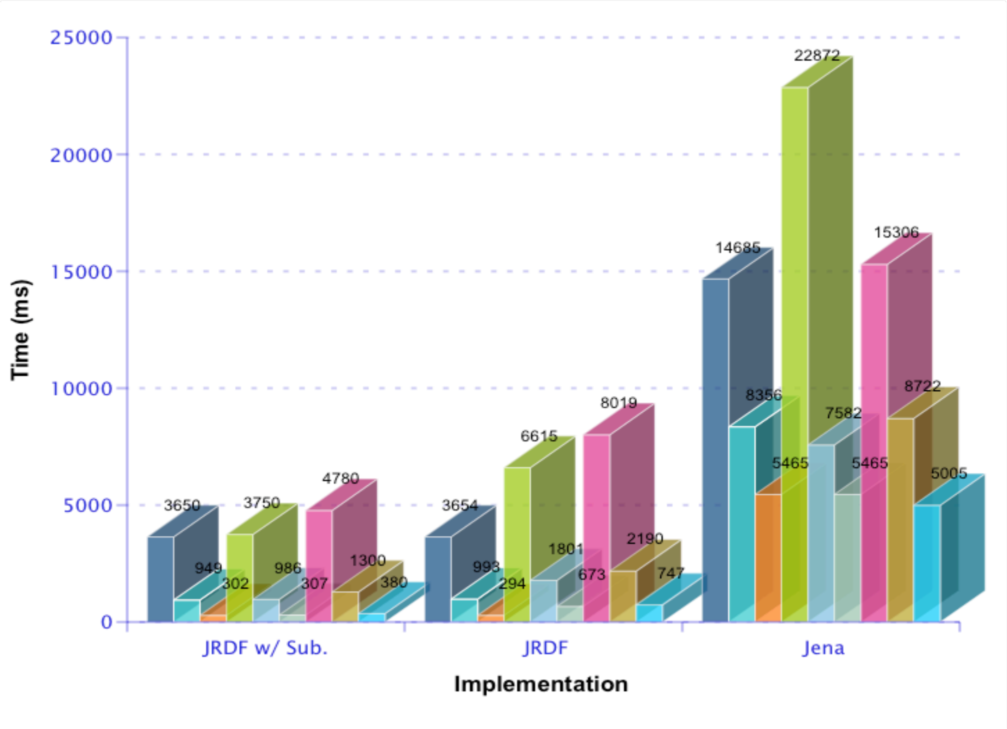

Figure 2, Query Performance of Various SPARQL Implementations

The difference in performance between the two implementations in JRDF can be quantified by comparing how many times faster “JRDF w/ Sub.” is compared to “JRDF”:

	~100K Statements	~50K statements	~25K statements
Query 2	1.76	1.82	2.19
Query 3	1.68	1.68	1.96

Query 1 uses only UNION and join (“.”) and so there are no significant differences between the two.

5 Future Work

5.1 Extending the use of the Relational Model

The relational model offers the ability to simplify and extend SPARQL. For example, the two SPARQL operations: CONSTRUCT and ASK are designed to produce two alternate result forms. The first returns an RDF graph and the second returns a true/false result depending on if the query matches the given constraints. The creation of new operations is not required and is suitably supported by the relational model.

The relational model is capable of returning results sets that allow the reconstruction of the original graph or to create a new one based on the results returned. Likewise, with the introduction of the true and false relation, TABLE_DEE and TABLE_DUM [6] respectively, SPARQL's ASK operation can be performed by performing a project over no columns. If there are results returned a project over no columns will produce TABLE_DEE or true.

The set-based operation SUMMARIZE provides the same functionality as SQL's 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)." [6].

The SUMMARIZE operation can also be used to generate a new relation that represents a graph based on the results of a query. Each row will produce new entries in the graph.

Some examples of results being turned into graphs include:

?s1:subject	?s2:subject	?p:predicate	object:?o1	object:?o2
A	b	c	d	e
A	c	c	d	e

Produces the set of statements: {a, c, d} {b,c,d} {a,c,e} {b,c,e} {c, c, d} {c,c,e}

?foo:URI	?bar:URI
a	b

Produces the set of statements: {a, a, a} {a, a, b} {a, b, a} {a, b, b} {b, a, a} {b, a, b}, {b,b,a}, {b,b,b}

Finally, the use of nested relations [6] may off a more efficient way of returning results than currently provided [3].

5.2 Further Optimisations

If compositional semantics is chosen for SPARQL then many other optimisation techniques could be applied. These include the original System-R optimisation but also some of the more recent attempts to optimise and distribute left outer join and antijoin queries.

As noted, some of the optimizations given in Galindo-Legaria's work [13] have not been implemented. Treating relations that contain null values as a collection of non-null relations may be used to implement the given simplifications. The effect of maintaining many relations may, however, reduce the advantages of this approach.

Many of these optimisation techniques require the creation of query graphs which themselves could be implemented using RDF. Given the right operations SPARQL queries could be performed to find optimal query paths and could be useful in other domains.

The original work performed by Galindo-Legaria [13] has been criticised as being able to handle only simplistic queries where queries that require Cartesian products or predicates that refer to more than one table [22, 24]. To improve the instances where outer joins, inner joins an antijoins can be optimized [24]:

> *"the normal eligibility list of each predicate with some additional information and use it to determine the correct join order...An extended eligibility list (EEL) of a predicate p includes all the tables needed as the input to p in order to get the correct answer"*

This approach allows the efficient handling of multiple conjuncts, hyper-predicates and Cartesian products and could be applied to SPARQL optimisation.

5.3 A Minimum Union version of OPTIONAL Using SQL

While SQL is not a natural match to RDF it is a widely deployed platform that has proven scalability and performance characteristics and remains a key piece of infrastructure used in organisations. With this in mind it is important to consider how this work can be applied to SQL databases.

Existing work translating SPARQL to SQL [2] did not apply a minimum union approach to implementing OPTIONAL. Larson and Zhou have applied the Galindo-Legaria approach of translating queries into join-disjunctive form, where outer join queries are implemented as outer unions, using SQL for view matching [19]. The results from this work have shown that the outer join version of the queries outperformed existing approaches and could subsequently be applied to an SQL version of OPTIONAL.

6 Conclusion

Even at this early stage of development, research has highlighted the requirement to apply a formal model to SPARQL. The benefits of a formal model have been shown in previous query languages and it is apparent that SPARQL would also benefit from having a formal model. An applicable formal model must follow a number of explicit requirements including being compatible with the RDF data model.

An appropriate formal model, the relational model, was used to implement parts of the RDF querying language, SPARQL. Previous works on optimisation techniques, based on the relational model, were used to efficiently implement the SPARQL operations OPTIONAL and UNION. An order independent version of OPTIONAL was also considered in order to provide a possible alternative to the current semantics of SPARQL's OPTIONAL.

Relational optimisation techniques have been employed over a considerable period of time and constitute a significant amount of existing work. It has been shown that some of these techniques can successfully be applied to SPARQL but more work is needed. These techniques are reliant on a method of bottom-up or compositional evaluation that is not currently supported by SPARQL. The negative effect, of not applying this method of evaluation, would be to hamper intuitiveness and reduce the potential for future optimisation techniques.

References

[1] J. Bailey et al., “Web and Semantic Web Query Languages: A Survey,” LNCS 3564, 2005, Norbert Eisinger, Jan Maluszynski (editor(s)),

[2] A. Chebotko et al., *Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns*, Department of Computer Science, 2006; http://www.cs.wayne.edu/~artem/main/research/TR-DB-052006-CLJF.pdf

[3] K. Clark, *SPARQL Protocol for RDF*, World Wide Web Consortium (W3C) Candidate Recommendation, 2006; http://www.w3.org/TR/rdf-sparql-protocol/

[4] E. F. Codd, “A Relational Model for Large Shared Databanks,” Communications of the ACM, vol. 13, no. 6, 1970, pp. 377–387.

[5] R. Cyganiak, *A Relational Algebra for SPARQL*, Digital Media Systems Laboratory, HP Laboratories Bristol, Tech. Rep, HP Laboratories Bristol, Tech. Rep, 2005; http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html

[6] C. J. Date, *Database in Depth, Relational Theory for Practitioners*, O’Reilly Media, Inc, Sebastopol, California, 2005, pp. 4–10, 41–57, 86–93.

[7] C. J. Date, *A Sweet Disorder*, 2004; http://www.dbdebunk.com/page/page/649881.htm

[8] C. J. Date, and H. Darwen, *Databases, Types, and the Relational Model, The Third Manifesto*, Addison Wesley Longman, Inc, Reading, MA, USA, 2006.

[9] M. David, “ANSI SQL hierarchical processing can fully integrate native XML,” SIGMOD Rec., Vol. 32, No. 1, 2003, ACM Press, pp. 41–46.

[10] F. Frasincar et al., “RAL: An Algebra for Querying RDF,” World Wide Web, Internet and Web Information Systems (WWW), vol. 7, no. 1, 2004, pp. 83–109.

[11] C. Galindo-Legarai, “Outerjoins as Disjunctions,” Proceedings of the 1994 ACM-SIGMOD Int. Conference on Management of Data, 1994, pp. 348 – 358.

[12] C. Galindo-Legaria, *Algebraic Optimization of Outerjoin Queries*, Doctoral dissertation, Harvard University, University, Cambridge, Massachusetts, 1992.

[13] C. Galindo-Legaria, and A. Rosenthal, “Outerjoin Simplification and Reordering for Query Optimization,” ACM Transactions on Database Systems, Vol. 22, No. 1, 1997, pp. 43–74.

[14] P. Haase et al., “A Comparison of RDF Query Languages,” Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[15] S. Harris, *DAWG Testcases*, World Wide Web Consortium (W3C), 2006; http://www.w3.org/2001/sw/DataAccess/tests/

[16] S. Harris, and N. Shadbolt, “SPARQL Query Processing with Conventional Relational Database Systems,” WISE 2005 International Workshops, New York, NY, USA, 2005, pp. 235–244.

[17] 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, 2003.

[18] G. Klyne, J. Carroll, and A. Seaborne, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, World Wide Web Consortium (W3C) Recommendation, 2004; http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[19] P. Larson and J. Zhou, "View Matching for Outer-Join Views," Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005, pp. 445–456.

[20] L. Libkin, and L. Wong, "Query Languages for Bags and Aggregate Functions," Journal of Computer and System Sciences, vol. 55, no. 2, 1997, pp. 241–272.

[21] A. Pamukci, *Outer Join in Edutella*, Master's thesis, Stockholm University, 2004.

[22] J. Perez, M. Arenas, and C. Gutierrez, *Semantics and Complexity of SPARQL*, 2006; http://arxiv.org/abs/cs.DB/0605124

[23] E. Prud'hommeaux, and A. Seaborne*, SPARQL Query Language for RDF*, World Wide Web Consortium (W3C) Candidate Recommendation, 2006; http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/

[24] J. Rao et al., "Using EELs, a practical approach to outerjoin and antijoin reordering," 00, Los Alamitos, CA, USA, 2001, IEEE Computer Society, pp. 585–594.