*I.J. Information Engineering and Electronic Business*, **2015, 2, 20-26**
Published Online March 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2015.02.04

# An Analysis of RDF Storage Models and Query Optimization Techniques

**Asim Sinan Yuksel and Ibrahim Arda Cankaya**
Suleyman Demirel University Computer Engineering Department, E9 Building, West Campus, Isparta, 32200, Turkey
Email: {asimyuksel, ardacankaya}@sdu.edu.tr

**Mehmet Erkan Yuksel**
Istanbul University Computer Engineering Department, Avcilar, Istanbul, 34320, Turkey
Email: eyuksel@istanbul.edu.tr

***Abstract*—**The Web provides access to substantial amount of information. Metadata that means data about data enables the discovery of such information. When the metadata is effectively used, it increases the usefulness of the original data/resource and facilitates the resource discovery. Resource Description Framework (RDF) is a basis for handling these metadata and is a graph-based, self-describing data format that represents information about web-based resources. It is necessary to store the data persistently for many Semantic Web applications that were developed on RDF to perform effective queries. Because of the difficulty of storing and querying RDF data, several storage techniques have been proposed for these tasks. In this paper, we present the motivations for using the RDF data model. Several storage techniques are discussed along with the methods for optimizing the queries for RDF datasets. We present the differences between the Relational Database and the XML technology. Additionally, we specify some of the use cases for RDF. Our findings will shed light on the current achievements in RDF research by comparing the different methodologies for storage and optimization proposed so far, thus identifying further research areas.

***Index Terms*—**Resource Description Framework, RDF Storage Models, RDF Query Languages, RDF Use Cases, Query Optimization Techniques, Semantic Web.

## I. Introduction

Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) standard that represents information about resources on the web [1]. RDF employs the idea of using web identifiers and descriptions of resources in terms of simple attributes and their values [2]. Originally designed as a metadata model, that represented metadata about web resources such as the title, author, copyright and licensing information, RDF has evolved into a more expansive concept with the generalization of the concept of “web resources”. It is used to identify the resources on the web rather than just retrieve it [3]. This equips RDF to represent the information that can be processed by the applications and not just be displayed on search. By providing a common framework, it provides exchange of information between the applications without loss of meaning. RDF is a key to the implementation of ‘Semantic Web’ activity proposed by the W3C, the next evolutionary stage of the internet activity enhancement where the automated programs can store, exchange and make use of the machine-readable information located throughout the Web, making the information handling activities on the Web more efficient [3]. RDF was first published as a data model with the XML syntax as a W3C Recommendation in 1999 [1] and the newer, improved version of RDF was later published in 2004. Since then, there has been a growing interest in exploiting the benefits proposed by the RDF model and our paper sets the roadmap of the different contributions to the RDF data model proposed so far. We organize the paper as follows. In Section II, we first present a review of the RDF data model. In Section III, IV, its comparison to XML and Relational Database are covered. In Section V, some of the popular use cases are presented. In Section VI, we discuss the need for efficient storage models and describe some of the models that have been proposed so far. In Section VII, the evolution of the different query languages is presented with the emphasis on SPARQL. In Section VIII, we investigate the considerations for optimizing queries on these storage models and present SPARQL. Additionally, we investigate different optimization techniques, and finally in Section IX and X, we present our findings and conclude with an overview of possible future research directions.

## II. RDF Data Format

The RDF data model draws from the concepts of Relational DBMS (RDBMS) and uses a conceptual modeling approach similar to the Entity-Relationship [4]. It makes use of the subject-predicate-object format of expressions to describe Web resources. These statements are called ‘triples’ in RDF terminology. The RDF statement has a Uniform Resource Identifier (URI) as its subject. A predicate is a URI implying a relationship and the object may be a URI or Unicode string literal [4]. The abstract models of RDF have several serialization or file formats and the triples can be encoded in one of several of these formats. The most common serialization format is by using the XML syntax to write and exchange RDF graphs, referred to as RDF/XML implementations. Other serialization formats include the Notation 3 or N3, a non-XML implementation that is purported to be easier to follow and written by hand [4]. It is based on the tabular notation that makes the triples easier to recognize. N3 is similar to the Turtle and N-Triples formats [4]. RDF also supports the SPARQL Protocol and the RDF Query Language (SPARQL) for RDF graphs. The language was published as a W3C recommendation in January 2008 compared to other query languages like RDQL, Versa, RQL and XUL [2].

## III. RDF VS. RELATIONAL DATABASE

It may be seen that RDF is also a relational database, however the idea behind it is not the same as the idea behind the relational model. RDF statements consist of the subject, predicate, object that is called triples. The most important aspect of RDF is that the triples are identified with URIs. This means that they play the role of key fields such as the primary keys or IDs in the local storage. Compared to the relational database, the Web can be seen as a huge single database. In the relational model, a row represents a single data item in a table. A SELECT query is a filter that selects data from a database. A relational database is a storage system that represents the information in tables, rows and columns. A significant difference between the relational databases and the RDF is that a relation is true when there is a matching row in the table or it is false when there is no value returned. However, in RDF, if the values are not in the “row”, it is not false but it is an unknown value.

## IV. RDF VS. XML

RDF and XML are two existing standards for representing the data on the Web. In the case of XML, it basically addresses the document structure, while in the case of RDF, it provides a “data model that can be extended to address sophisticated ontology representation techniques” [6]. There are several reasons for using RDF rather than XML.

Firstly, processing XML requires that the closing element tag be reached for the processing to be complete. For instance, if an XML document is parsed into elements in the memory, the end of the same element should be processed before transferring all the elements and nested parts into another persisted form of data. Thus, all the elements that contain other elements are fetched into memory until their data members are processed. This increases the memory usage, especially with the large XML documents. However, RDF allows the processing of the first element quickly because the data is stored in another element in the same document. It becomes easy for an application to reconstruct the original data since the same URI establishes the relationship between these two elements.

Another benefit of RDF is that querying the data requires only knowledge of the triple structure. In XML, the user needs to provide the entire structure of all of the elements, that is the full path, in order to retrieve the proper value. The entire document has to be traversed to answer this query, making it very cumbersome and hard to work with. Additionally, with RDF, it is possible to join the data from two different sources even if there are structural differences between them. This feature plays an important role in the business world where there is a need to share and combine the data from different sources.

## V. RDF USE CASES

Currently, RDF is used in the offline knowledge management applications. Other popular applications of RDF include Really Simple Syndication (RSS), Friend of a Friend (FOAF) [18] that describes people with common interests and interconnections, Haystack client [19], a semantic web browser developed by MIT and MusicBrainz [20] that publishes the information about music albums. In addition, there are several other applications of RDF that rely on its flexibility in sharing vocabularies.

Chandler [21] is a personal information management application, which is an open source technology and includes RDF and its specifications. RDF Gateway [22] is an integrated web server and database that uses RDF/XML. Siderean Seamark [23] is a versatile application, which provides resources for intelligent site querying and navigation. Adobe, a major player in the graphics industry, is one of these companies. Its RDF strategy is known as XMP [24]. XMP focuses on providing a metadata label that can be embedded directly into the applications, files, and databases, including binary data.

## VI. STORAGE MODELS

Various storage schemes have been proposed for RDF Data Storage. They can be classified into three main categories: File system, RDF storage using RDBMS and RDF storage using OODBMS.

### *A. File System*

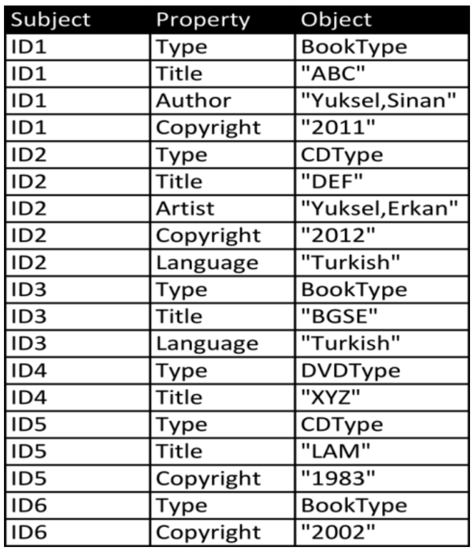
Currently, applications on the web make use of XML notations to store metadata about web pages. This type of storage model typically requires an XML parser that can extract the different elements from the XML file in order to obtain the information regarding RDF triples namely subjects, predicates or objects.

### *B. RDF Storage Using RDBMS*

RDF data consists of triples - a subject, a predicate and an associated object. Consequently, most studies have focused on this property of the RDF data model to propose different ways to store RDF data. Typically, different storage methods that make use of RDBMS have concentrated on using the relational model to efficiently store and query different RDF data as described below.

### *B.A Triplestore*

This is a simple three-column table in RDBMS where three columns have the attributes as subject, property and object [5]. Fig. 1 shows the triple storage example. Generally these tables are very huge. This type of storage is efficient when a large amount of data is retrieved from the table, however it is not appropriate when retrieving very small amounts of data. This is because in the latter case, large amounts of data will have to be scanned in order to retrieve a small portion of it. Another interesting feature of storing data as triples is that it just requires self-joins in order to answer queries, since all data is stored in a single table in the former.

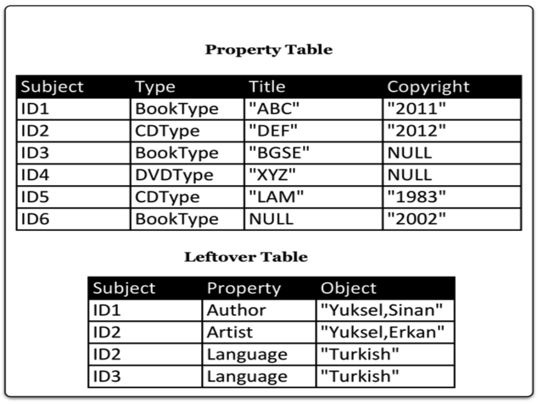


| Subject | Property | Object |
|---|---|---|
| ID1 | Type | BookType |
| ID1 | Title | "ABC" |
| ID1 | Author | "Yuksel,Sinan" |
| ID1 | Copyright | "2011" |
| ID2 | Type | CDType |
| ID2 | Title | "DEF" |
| ID2 | Artist | "Yuksel,Erkan" |
| ID2 | Copyright | "2012" |
| ID2 | Language | "Turkish" |
| ID3 | Type | BookType |
| ID3 | Title | "BGSE" |
| ID3 | Language | "Turkish" |
| ID4 | Type | DVDType |
| ID4 | Title | "XYZ" |
| ID5 | Type | CDType |
| ID5 | Title | "LAM" |
| ID5 | Copyright | "1983" |
| ID6 | Type | BookType |
| ID6 | Copyright | "2002" |

Fig. 1 Triple store example.

### *B.B Clustered Property Table*

By applying the clustering property table technique, RDF tables are de-normalized by storing them in a flattened, wider representation that is similar to the traditional relational schemas [5]. Fig. 2 illustrates the clustered property table example. Flattening operation is done by finding the sets of properties that are grouped together. This table requires less number of joins to access because it eliminates the self-joins on the subject.



**Property Table**

| Subject | Type | Title | Copyright |
|---|---|---|---|
| ID1 | BookType | "ABC" | "2011" |
| ID2 | CDType | "DEF" | "2012" |
| ID3 | BookType | "BGSE" | NULL |
| ID4 | DVDType | "XYZ" | NULL |
| ID5 | CDType | "LAM" | "1983" |
| ID6 | BookType | NULL | "2002" |

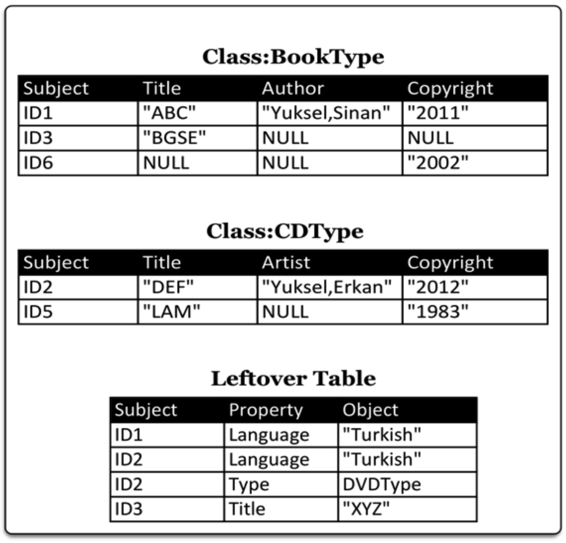
**Leftover Table**

| Subject | Property | Object |
|---|---|---|
| ID1 | Author | "Yuksel,Sinan" |
| ID2 | Artist | "Yuksel,Erkan" |
| ID2 | Language | "Turkish" |
| ID3 | Language | "Turkish" |

Fig. 2. Clustered property table example.

### *B.C Property Class Table*

The property-class table clusters similar groups of subjects by using the type of subjects. Additionally, properties may exist in multiple property-class tables as shown in Fig. 3. In Jena2 [6], the property-class tables are used to store the reified statements. Oracle also adopted a data structure similar to the property class table that is called “subject-property matrix” to increase the speed of RDF queries. The property table approach has advantages over the triple-store since it reduces joins of the tables. An example table is shown in Fig. 3.



**Class:BookType**

| Subject | Title | Author | Copyright |
|---|---|---|---|
| ID1 | "ABC" | "Yuksel,Sinan" | "2011" |
| ID3 | "BGSE" | NULL | NULL |
| ID6 | NULL | NULL | "2002" |

**Class:CDType**

| Subject | Title | Artist | Copyright |
|---|---|---|---|
| ID2 | "DEF" | "Yuksel,Erkan" | "2012" |
| ID5 | "LAM" | NULL | "1983" |

**Leftover Table**

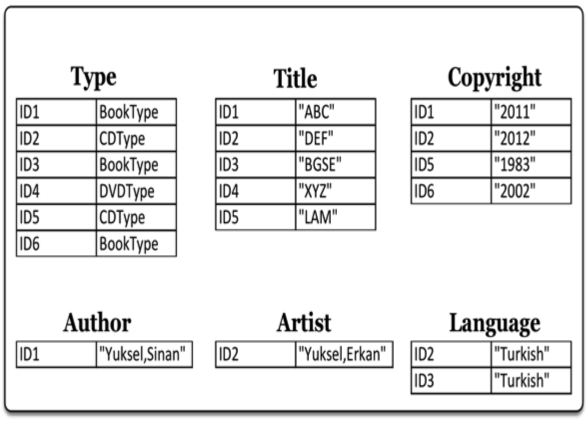
| Subject | Property | Object |
|---|---|---|
| ID1 | Language | "Turkish" |
| ID2 | Language | "Turkish" |
| ID2 | Type | DVDType |
| ID3 | Title | "XYZ" |

Fig. 3. Property class table example.

### *B.D Vertical Partitioning*

Vertical partitioning [5] can be defined as creating two column tables based on the unique properties, with one column representing subjects and the second column referring to the objects as shown in Fig. 4. Advantages of the vertical partitioning approach over the previous approach are:

- **Support for Attributes with Multiple Values:** When a particular property has more than one value for a subject, every unique value for that particular property is listed in a row of the table.
- **Heterogeneous Record Support:** Subjects that are defined without a particular property are skipped from the table for that property. Therefore, this avoids the explicit storage of NULL data. When the data is not well defined, this feature becomes useful.
- **Fewer Unions and Fast Joins:** The property table approach reduces the need for union clauses in the queries since whole data is present in the same table. Moreover, in the vertical partitioning approach, there are more joins than the property table approach. However, the properties are joined using simple, fast join algorithms that make the vertical partitioning approach more preferable than the property table approach.



**Type**

| | |
|---|---|
| ID1 | BookType |
| ID2 | CDType |
| ID3 | BookType |
| ID4 | DVDType |
| ID5 | CDType |
| ID6 | BookType |

**Title**

| | |
|---|---|
| ID1 | "ABC" |
| ID2 | "DEF" |
| ID3 | "BGSE" |
| ID4 | "XYZ" |
| ID5 | "LAM" |

**Copyright**

| | |
|---|---|
| ID1 | "2011" |
| ID2 | "2012" |
| ID5 | "1983" |
| ID6 | "2002" |

**Author**

| | |
|---|---|
| ID1 | "Yuksel,Sinan" |

**Artist**

| | |
|---|---|
| ID2 | "Yuksel,Erkan" |

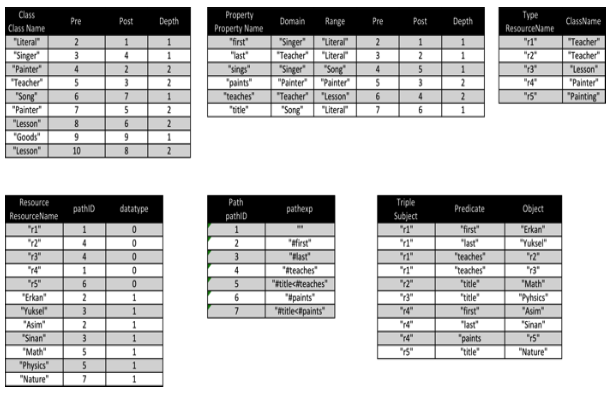
**Language**

| | |
|---|---|
| ID2 | "Turkish" |
| ID3 | "Turkish" |

Fig. 4. Vertical partitioning example.

## *B.E Path Based Storage*

Ref. [7] proposes a path based relational RDF storage. This storage scheme parses the RDF data and generates its own RDF graph. This RDF graph is decomposed into sub graphs that are then stored in the distinct relational tables by applying specific techniques. The decomposition into sub-graphs is based on the type of the predicate. These sub-graphs are based on Class Inheritance, Property Inheritance, Type information, Domain-range and the remaining data fall into Generic graphs. Based on the sub-graphs, [7] designed their relational schema that includes the relations class, property, resource, triple, path and type as shown in Fig. 5. Since this storage scheme retains the schema information and the path expression for each data source, it makes it possible to process the path based queries efficiently.



| Class<br>Class Name | Pre | Post | Depth |
|---|---|---|---|
| "Literal" | 2 | 1 | 1 |
| "Singer" | 3 | 4 | 1 |
| "Painter" | 4 | 2 | 2 |
| "Teacher" | 5 | 3 | 2 |
| "Song" | 6 | 7 | 1 |
| "Painter" | 7 | 5 | 2 |
| "Lesson" | 8 | 6 | 2 |
| "Goods" | 9 | 9 | 1 |
| "Lesson" | 10 | 8 | 2 |

| Property<br>Property Name | Domain | Range | Pre | Post | Depth |
|---|---|---|---|---|---|
| "first" | "Singer" | "Literal" | 2 | 1 | 1 |
| "last" | "Teacher" | "Literal" | 3 | 2 | 1 |
| "sings" | "Singer" | "Song" | 4 | 5 | 1 |
| "paints" | "Painter" | "Painter" | 5 | 3 | 2 |
| "teaches" | "Teacher" | "Lesson" | 6 | 4 | 2 |
| "title" | "Song" | "Literal" | 7 | 6 | 1 |

| Type<br>ResourceName | ClassName |
|---|---|
| "r1" | "Teacher" |
| "r2" | "Teacher" |
| "r3" | "Lesson" |
| "r4" | "Painter" |
| "r5" | "Painting" |

| Resource<br>ResourceName | pathID | datatype |
|---|---|---|
| "r1" | 1 | 0 |
| "r2" | 4 | 0 |
| "r3" | 4 | 0 |
| "r4" | 1 | 0 |
| "r5" | 6 | 0 |
| "Erkan" | 2 | 1 |
| "Yuksel" | 3 | 1 |
| "Asim" | 2 | 1 |
| "Sinan" | 3 | 1 |
| "Math" | 5 | 1 |
| "Physics" | 5 | 1 |
| "Nature" | 7 | 1 |

| Path<br>pathID | pathexp |
|---|---|
| 1 | "" |
| 2 | "#first" |
| 3 | "#last" |
| 4 | "#teaches" |
| 5 | "#title<#teaches" |
| 6 | "#paints" |
| 7 | "#title<#paints" |

| Triple<br>Subject | Predicate | Object |
|---|---|---|
| "r1" | "first" | "Erkan" |
| "r1" | "last" | "Yuksel" |
| "r1" | "teaches" | "r2" |
| "r1" | "teaches" | "r3" |
| "r2" | "title" | "Math" |
| "r3" | "title" | "Pyhsics" |
| "r4" | "first" | "Asim" |
| "r4" | "last" | "Sinan" |
| "r4" | "paints | "r5" |
| "r5" | "title" | "Nature" |

Fig. 5. Path based storage example.

## *B.F RDF Storage Using OODBMS*

RDF data can also be stored as a graph in an object oriented database [7] or semi-structured database. RDF storage system maps RDF graph structure onto its storage structure to support RQL. In these systems, data is stored as triples in RDB. The graph is built from triples to evaluate RQL queries on those triples.

In the graph model, RDF statements are represented as nodes and edges where nodes are either resources or values. This graph design simplifies the storage concept and has several advantages:

- It is possible to store the graph without reorganizing it. Therefore, storage design becomes simpler.
- Graph can be interpreted directly as it is already in the storage. No external mapping is required.

## VII. Evolution of RDF Query Languages

Fig. 6 shows the evolution of the different query languages proposed for the RDF data model. During the past decade, there have been several proposals made for an efficient query language for RDF data. However, none of them could satisfy sufficient criteria to be widely accepted. Table 1 illustrates the different desirable features of a query language and some examples of earlier proposed query languages.

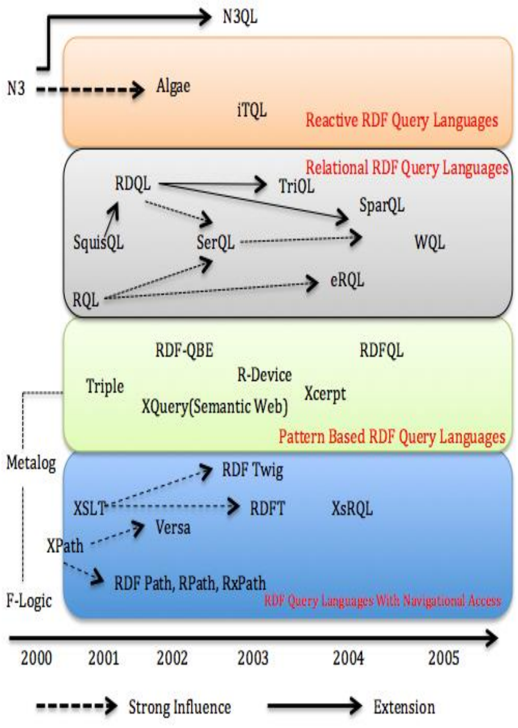


Fig. 6. From [9] Chronological overview of RDF query languages.

Among these proposals, SPARQL was the main focus of many researchers in order to enhance its expressiveness in treating RDF data such that it finally became a W3C standard in 2008. Hence, current research mostly focuses on developing query optimization techniques using SPARQL.

Table 1. Comparison between different query languages proposed before SPARQL

| Properties → / ↓ Query Languages | Algebric Functions | Quantification | Difference | Union | Sorting | Count | Grouping |
|---|---|---|---|---|---|---|---|
| RQL | YES | YES | YES | YES | NO | YES | NO |
| SeRQL | YES | NO | NO | NO | NO | NO | NO |
| RDQL | YES | NO | NO | NO | NO | NO | NO |
| TRIPLE | YES | NO | NO | YES | NO | NO | NO |
| N3 | YES | NO | NO | YES | NO | YES | NO |
| VERSA | YES | NO | NO | YES | NO | YES | NO |
| SPARQL | YES | YES | YES | YES | YES | YES | YES |

## VIII. Query Evaluation and Optimization Techniques

From the past thirty years, DBMS has undergone several changes and still research is being carried out in the area of optimization. This elevates the importance of query optimization in increasing the performance of the queries. The basic idea behind the query optimization is to find the optimal plan for query execution. There are several ways that optimization can be done. These include the use of indices, consideration for the semantics of the RDF data, estimation of the selectivity of triple and join patterns and finally the conversion of SPARQL queries into the SQL queries.

### *A. Use of the Indices*

Use of the indices have been successfully proposed and implemented in the relational model. To achieve similar performance, several indices are proposed based on the way the RDF data is stored. For instance, [10] proposed the use of B-tree index when the data is stored in two tables, one of which stores all the triples, and the other table stores only the corresponding URIs. The B-tree index is built on either the subject or the predicate.

Ref. [11] used B-tree indices when four tables (class, sub-class, property and sub-property) are used to store the RDF data. In this case, B-tree indices are built on subject, predicate and object.

Ref. [12] employed B+ tree index on a single triple table, by considering all six combinations of triples as well as any frequently used projections, making the index compressed that reduces the amount of space utilization. This method works well when the RDF data is stored in triple table.

Furthermore, [13] combined three indexes to propose the mixed index structure that consists of a B+ tree, Path index, and Context index, which was found to work well for both RDF graphs as well as RDF tables, and which increased the performance of long path queries.

Ref. [14] proposed the GRIN index, which is a balanced tree data structure optimized for RDF graph storage model. This was found to increase the performance of graph based queries.

Ref. [15] proposed the Triple-T index, which consists of B+ tree index built on all combinations of triples. This was found to be very efficient for RDF data stored in one single triple table.

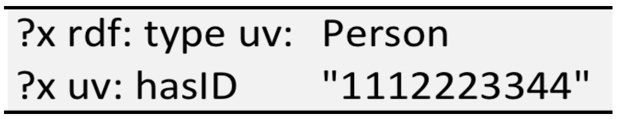
### *B. Considerations for the Semantics of Data*

In the relational database, the schema holds the information regarding the relationships among the data, primary and foreign key that can be used to optimize the queries. However, the RDF schema does not provide such facilities, making it necessary to explicitly state any constraints. In case the relational data is desired for the semantic web, it should be mapped to RDF without losing any of these constraints.

According to [16], the consideration of primary key and foreign key is important since it leads to the semantic optimization by restricting the number of valid states in the graph. Therefore, [16] suggested that any SPARQL query can be rewritten for the optimization by following the sequence of operations such as replacing the operators, eliminating the redundant joins if possible and reordering the remaining joins. This technique considerably increased the performance of queries for the RDF data stored in memory.

### *C. By Selectivity Estimation*

Using selectivity estimation, SPARQL queries are optimized based on the selectivity of basic graph patterns (BGP). Consider the case where the data is stored in memory [17]. We can achieve join order optimization by knowing the selectivity of the triple pattern and join the triple patterns from the statistics of the dataset that we are considering. Consider the two triple patterns shown in Fig. 7. If we have sufficient statistics, we can state that second triple pattern should be executed first because its result set is considerably small when compared to the second triple pattern. By reordering the queries, we can therefore rewrite the SPARQL queries that increase the performance. If the queries have joins, we need to estimate the selectivity of join triple patterns along with the above.

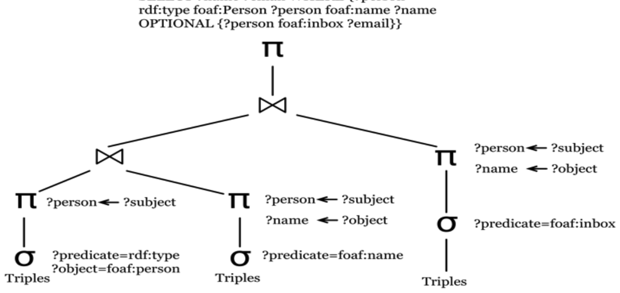


| | |
|---|---|
| ?x rdf: type uv: | Person |
| ?x uv: hasID | "1112223344" |

Fig. 7. Triple pattern.

### *D. By Converting SPARQL into Relational Algebra*

Relational Algebra is an intermediate language for the expression and the analysis of the queries that is widely used in the DBMS. A query represented in the relational algebra helps the query engine to perform better. Similarly, considering RDF data stored in tables, the SPARQL query can be represented in the relational algebra. [18] proposed a relational algebra for SPARQL that explains all the operations performed on the RDF relations. They also explain how the SPARQL query can be represented in the relational algebra that is shown in Fig. 8.



```
SELECT ?name ?email WHERE {?person
rdf:type foaf:Person ?person foaf:name ?name
OPTIONAL {?person foaf:inbox ?email}}
```

π
⋈
⋈
π ?person ← ?subject ?name ← ?object
π ?person ← ?subject
σ ?predicate=rdf:type ?object=foaf:person
Triples
π ?person ← ?subject ?name ← ?object
σ ?predicate=foaf:name
Triples
σ ?predicate=foaf:inbox
Triples

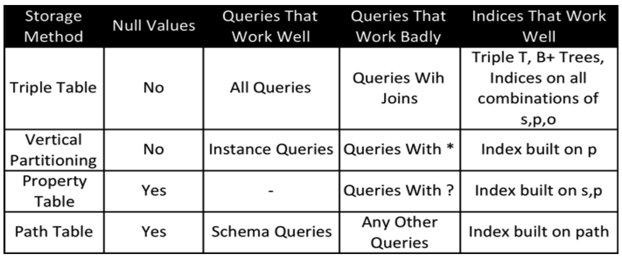
Fig. 8. Representation of SPARQL query in relational algebra.

Different operators like selection, projection and rename; inner-join and left outer-join and union of relational algebra can be mapped to the operations of SPARQL that can be further mapped to SQL. By doing this, we can extend all the optimization techniques of SQL to SPARQL. During optimization, we may come across some mismatches during this conversion. In relational algebra, the missing variable is represented as NULL. However, in SPARQL, it is represented as UNBOUND variable. Relational algebra rejects the tuple combination when there is a NULL value in the join attribute. However, in RDF relational algebra, the tuple combination is rejected only when an attribute is bound on both sides with two different values. Operators such as OPTIONAL and FILTER cannot be represented in the relational algebra.

## IX. Results

In this study, we have identified that triple table storage is simple to construct; however it requires a lot of self joins to retrieve data. Vertical partitioning works well when there are less number of properties involved. On the other hand, this requires more space. Property table approach works well with structured data and reduces the number of self-joins when compared to triple table storage. When dealing with unstructured data, this approach requires insertion of null values into a table that is not desirable. Path-based storage performs better for schema based queries. Nevertheless, the overhead with this approach is the creation of tables. Fig. 9 compares the benefits and drawbacks of the different storage models encountered.

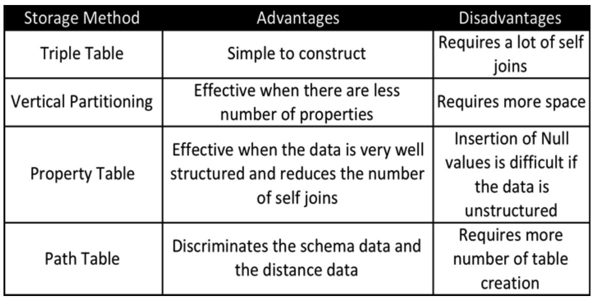
By further analysis, we see that triple table storage works well for all kinds of queries and also supports the Triple-T, B+ tree indexes. However, this storage model does not support null values and the model does not work well for queries that require more number of joins.



| Storage Method | Null Values | Queries That Work Well | Queries That Work Badly | Indices That Work Well |
|---|---|---|---|---|
| Triple Table | No | All Queries | Queries Wih Joins | Triple T, B+ Trees, Indices on all combinations of s,p,o |
| Vertical Partitioning | No | Instance Queries | Queries With * | Index built on p |
| Property Table | Yes | - | Queries With ? | Index built on s,p |
| Path Table | Yes | Schema Queries | Any Other Queries | Index built on path |

Fig. 9. Comparison of the benefits and drawbacks of different storage models.

Considering vertical partitioning, the instance queries are best supported. Furthermore, null values are not supported by the vertical partitioning and it does not scale well for queries with the * operator. The indexes built on the predicate are well supported. Although the property table supports null values, and provides good support for the index structures built upon subject and predicate, it does not perform well for queries with the “?” operator. The path-based storage is appropriate for schema queries, however it does not work well with any other query types, and does not support null values. For the path-based storage, the indexes built upon path are best supported. Fig. 10 summarizes our findings.



| Storage Method | Advantages | Disadvantages |
|---|---|---|
| Triple Table | Simple to construct | Requires a lot of self joins |
| Vertical Partitioning | Effective when there are less number of properties | Requires more space |
| Property Table | Effective when the data is very well structured and reduces the number of self joins | Insertion of Null values is difficult if the data is unstructured |
| Path Table | Discriminates the schema data and the distance data | Requires more number of table creation |

Fig. 10. Comparison of storage models.

## X. Conclusion

RDF is a data format that is claimed to be very attractive for achieving interoperability over the Web. However, the challenge remains to build scalable systems that provide efficient storage and query of the RDF data over a widely distributed network such as the Web. From our study, we clearly see that research in RDF is highly fragmented addressing different scenarios. We relate this problem to the lack of real world cases available that implement RDF on a large scale, and thus studies are mostly using hypothetical data.

We have also observed that different strategies for storing and querying the RDF data have been proposed, each with their own benefits and drawbacks. Although, benchmarks exist to compare the results, there is no consensus among researchers to compare their results on a common ground. We find this a serious limitation and suggest further investigation in that area. When compared to the relational model, we believe that RDF is still a very fertile area of research, especially when it comes to the best way to store and query the data that needs to be shared across different systems.

## References

[1] RDF Primer, http://www.w3.org/TR/rdf-primer/, Last Acces: 27.10.2014.

[2] Powers, S., *Practical RDF*, O'Reilly Media, 2003.

[3] Guha, R., McCool, R., Fikes, R., “Contexts for the Semantic Web”, The 3rd International Semantic Web Conference, Hiroshima, Japan, pp. 32-46, 2004. doi: 10.1007/978-3-540-30475-3_4.

[4] Abadi, D. J., Marcus, A., Madden, S. R., Hollenbach, K., “Scalable Semantic Web Data Management Using Vertical Partitioning”, The 33rd International Conference on Very Large Databases, Vienna, Austria, pp. 411-422, 2007.

[5] Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D., “Efficient RDF Storage and Retrieval in Jena2”, The 1st International Workshop on Semantic Web and Databases, Berlin, Germany, pp. 131-150, 2003.

[6] Matono, A., Amagasa, T., Yoshikawa, M., Uemura, S., “A Path-based Relational RDF Database”, The 6th Australasian Database Conference, Newcastle, Australia, pp. 95-103, 2005.

[7] Boenstroem, V., Hinze, A., Schweppe, H., “Storing RDF as a Graph”, The 1st Latin American Web Congress, Santiago, Chile, pp. 27-36, 2003.

[8] Reasoning web: second international summer school 2006, Lisbon, Portugal, September 4-8, 2006: tutorial lecture.

[9] Chong, E. I., Das, S., Eadon, G., Srinivasan, J., "An efficient SQL-based RDF Querying Scheme", The 31st International Conference on Very Large Data Bases, Trondheim, Norway, pp. 1216-1227, 2005.

[10] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K., "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases", The 2nd International Workshop on The Semantic Web, Hongkong, China, pp. 1-13, 2001.

[11] Thomas Neumann, Gerhard Weikum. RDF-3x: A RISC-style Engine for RDF. doi:10.14778/1453856.1453927

[12] Baolin, L. and Bo, H. Hprd: A high performance rdf database. In NPC, 2007. doi: 10.1007/978-3-540-74784-0_37.

[13] Octavian Udrea, Andrea Pugliese V.S. Subrahmanian. GRIN: A Graph Based RDF index.

[14] George H. L. Fletcher and Peter W. Beck, A role-free approach to indexing large RDF data sets in secondary memory for efficient SPARQL evaluation.

[15] Lausen, G., Meier, M., Schmidt, M., "SPARQLing Constraints for RDF", The 11th International Conference on Extending Database Technology, pp. 499-509, Nantes, France, 2008. doi:10.1145/1353343.1353404.

[16] Harth, A., Decker, S., "Optimized Index Structures for Querying RDF from the Web", The 3rd Latin American Web Congress, Buenos Aires, Argentina, pp. 71-80, 2005. doi:10.1109/LAWEB.2005.25.

[17] Cyganiak, R., "A Relational Algebra for SPARQL", 2005, http://www.hpl.hp.com/techreports/2005/HPL-2005-170.pdf, Last Access: 27.10.2014.

[18] Foaf Project, http://www.foaf-project.org, Last Access: 27.10.2014.

[19] Haystack, http://haystack.csail.mit.edu, Last Access: 27.10.2014.

[20] MusicBrainz, https://musicbrainz.org, Last Access: 27.10.2014.

[21] Chandler, http://blog.chandlerproject.org, Last Access: 27.10.2014.

[22] RDF Gateway, http://www.intellidimension.com, Last Access: 27.10.2014.

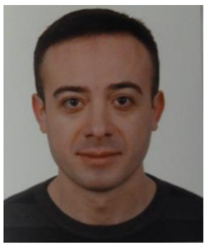
[23] Siderean Seamark, http://www.siderean.com, Last Access: 27.10.2014.

[24] XMP, http://www.adobe.com/products/xmp.html, Last Access: 27.10.2014.

## Authors' Profiles

**Asim S. Yuksel** obtained his B.S. degree in computer engineering from Ege University, Izmir, Turkey in 2006. He completed his Masters degree in computer science from Indiana University School of Informatics and Computing, Indiana, USA in 2010. He is currently a Ph.D. candidate in Istanbul University Department of Computer Engineering. He has been working as a research assistant in Suleyman Demirel University Department of Computer Engineering since 2012. His research interests include mobile security and privacy, social network security and privacy, object oriented programming and object oriented analysis and design.

**Ibrahim A. Cankaya** received his B.S. degree in 2012 from Suleyman Demirel University Computer Engineering Department, Isparta, Turkey. He continues his Master education at Suleyman Demirel University Computer Engineering Department, Isparta, Turkey. He has been working as a Research Assistant in Suleyman Demirel University Computer Engineering Department since 2012. His research interest includes algorithm and programming, database management, mobile programming.



**Mehmet E. Yuksel** obtained his B.S. degree in computer engineering from Firat University, Elazig, Turkey in 2005. He completed his Masters degree in computer engineering in 2010, his Ph.D. degree in 2014 from Istanbul University Computer Engineering Department, Istanbul. He worked as a research assistant in Istanbul University Computer Engineering Department between 2008 and 2012. His research interests include wireless sensor networks, cognitive networks, software defined networks, rfid and social networks.

**How to cite this paper:** Asim Sinan Yuksel, Ibrahim Arda Cankaya, Mehmet Erkan Yuksel,"An Analysis of RDF Storage Models and Query Optimization Techniques", IJIEEB, vol.7, no.2, pp.20-26, 2015. DOI: 10.5815/ijieeb.2015.02.04