On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs*

Sofia Alexaki Vassilis Christophides Greg Karvounarakis Dimitris Plexousakis
ICS-FORTH, Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece
{alexaki, christop, gregkar, dp}@ics.forth.gr

1 Introduction

The academic, corporate and industrial interest in information systems such as corporate memories, vertical aggregators, infomediaries, etc. enabling to select, classify and access, in a useful and meaningful way, various information resources (e.g., sites, documents, data) for diverse target audiences has been increasing over the last few years. These systems, termed *Community Web Portals*, rely on a *Catalog* holding descriptions, i.e., *metadata*, about the available resources on corporate intranets or the Web [6]. In order to effectively disseminate community knowledge, Portal Catalogs assimilate and organize information in a *multitude of ways*, which are far more flexible and powerful than those provided by standard (object, relational) or XML databases [3, 1]. Recent Web standards such as the W3C Resource Description Framework (RDF) [8, 5] are proved to be more suitable for creating and exchanging resource descriptions between Community Webs. In this paper, we address storage issues of voluminous RDF metadata using an object-relational DBMS (ORDBMS) and the catalog of the Open Directory Portal exported in RDF as test metadata.

RDF [8] provides i) a *Standard Representation Language* for metadata based on *directed labeled graphs* in which nodes are called *resources* (or *literals*) and edges are called *properties*, and ii) an *XML syntax* for expressing metadata in a form that is both humanly readable and machine understandable. The most distinctive RDF feature is its ability to support superimposed descriptions for the same Web resources, enabling content syndication - and hence, automated processing - in a variety of application areas (e.g., administration, recommendation, content rating, intellectual property rights, site maps, push channels, etc.). To interpret these descriptions within or across communities, RDF allows for the definition of schemas [5] i.e., vocabularies of labels for graph nodes (i.e., *classes*) and edges (i.e., *properties*) that can be used to describe and query *RDF description bases*. Many content providers (e.g., ABCNews, CNN, Time Inc.), Web Portals (e.g., Open Directory, CNET, XMLTree[1]), browsers (e.g., Netscape 6.0, W3C Amaya), as well as, emerging application standards for Web data and services syndication (e.g., the RDF Site Summary [4], Dublin Core [15], or the Web Service Description Language [16]) have already adopted RDF. In a nutshell, the growing number of Web resources and the proliferation of description services lead nowadays to large volumes of RDF metadata (e.g., the Open Directory Portal of Netscape comprises over 170M of Subject Topics and 700M of indexed URIs).

Storing Web data, such as RDF schema and resource descriptions, in an ORDBMS is not a novel issue. However, existing database representations and benchmarks [7, 13, 12, 14] need to be reassessed in our setting, since: a) RDF has a different data model featuring labels on both nodes and edges, as well as, taxonomies of labels (see Section 2), and, hence, demands a different querying functionality than in semistructured or XML databases [1]; b) our test metadata contain a significantly larger volume of schema information than that used in previous testbeds (i.e., XML elements, attributes). In this paper, we use PostgreSQL to compare the performance of two well-known database representations (*generic* versus *specific*), adapted to the peculiarities of RDF. More precisely, the schema-specific representation outperforms the generic representation both in the required storage volume and in query execution time. The performance of the two representations has been evaluated with a series of tests involving queries on schema, data or a combination thereof.

2 RDF by Example: The Open Directory Portal

In this section, we briefly recall the main modeling primitives proposed in the Resource Description Framework (RDF) Model & Syntax and Schema (RDFS) specifications [8, 5], using as example the catalog of Internet Portals, like Open Directory or Yahoo!. These catalogs use huge hierarchies of topics to semantically classify Web resources (e.g., in 15 out of ODP's 16 hierarchies 252825 topics are used and 1,770,781 sites are indexed). Additionally, various administrative metadata (e.g., titles, modification dates) of resources are usually created using a Dublin-Core like schema [15]. Then, users can either navigate through the topics of

*This work was partially supported by the EC project C-Web (IST-1999-13479) and Mesmuses (IST-2000-26074).
[1] www.dmoz.org, home.cnet.com, www.xmltree.com respectively.

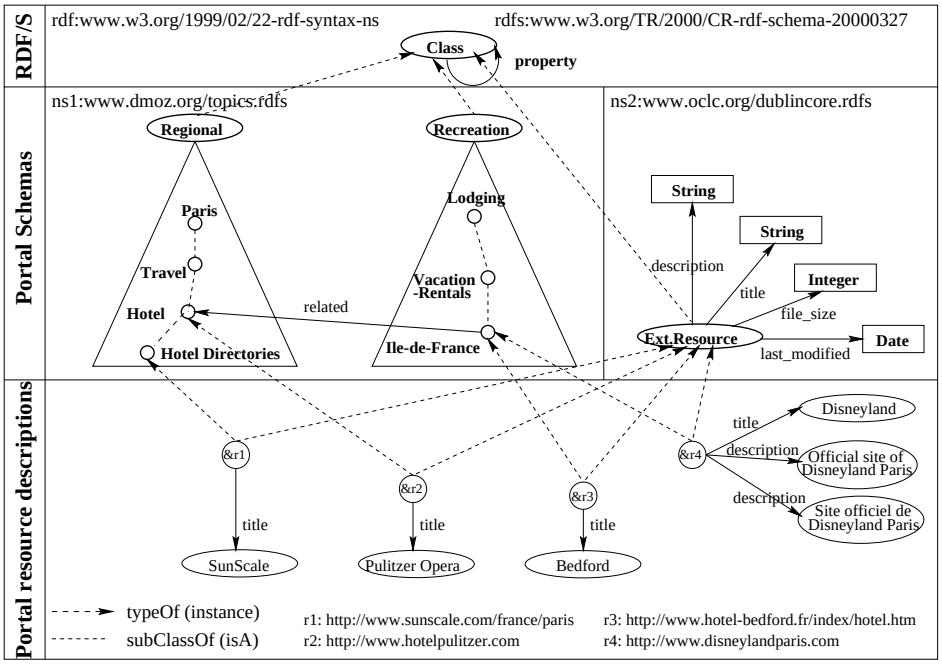

Figure 1: Modeling in RDF the Catalog of the Open Directory Portal

the catalog to locate resources of interest or issue a full-text query on topic names as well as the URIs and the titles of the resources. Portal Catalogs can be then easily and effectively represented using RDF/S.

The middle part of Figure 1 depicts the two schemas employed by ODP: the first is intended to capture the semantics of web resources while the second is intended for Portal administrators. The scope of the declarations is determined by the corresponding *namespace* definition of each schema, e.g., **ns1** (`www.dmoz.org/topics.rdfs`) and **ns2** (`www.oclc.com/dublincore.rdfs`). For simplicity, we will hereforth omit the namespaces as well as the paths from the root of the topic hierarchies prefixing class and property names (since topics have non-unique names). In the ODP topics schema, we can see two out of the sixteen hierarchies, namely, *Regional* and *Recreation* whose topics are represented as RDF classes (see the RDF/S default namespaces in the upper part of Figure 1). Various semantic relationships exist between these classes, either within a topic hierarchy (e.g., *subtopics*), or across hierarchies (e.g., *related* topics). The former, is represented using the RDF subclass relationship (e.g., *Travel* is a, not necessarily direct, subclass of *Paris*) and the latter using an RDF property named *related* (e.g., between the classes *Ile-de-France* and *Hotel*). Finally, the ODP administrative metadata schema contains various literal RDF properties such as *title*, *mime-type*, *last_modified*, defined on class *ExtResource*. Note that properties can also be organized in taxonomies in a manner similar to the organization of classes.

Using these schemas, we can see in the lower part of Figure 1, the descriptions created for four sites (resources `&r1-&r4`). For instance, `&r4` is a resource classified under both the classes *Ile-de-France* and *ExtResource* and has three associated literal properties: a *title* property with value "Disneyland" and two *description* properties with values "Official site of Disneyland Paris" and "Site officiel de Disneyland Paris" respectively. In the RDF jargon, a specific resource (i.e., node) together with a named property (i.e., edge) and its value (i.e., node) form a *statement*. Each statement can be represented by a *triple* having a *subject*, a *predicate*, and an *object* (e.g., `&r4`, `title`, and "Disneyland " respectively). The subject and object should be of a class compatible (under specialization) with the domain and range of the predicate (e.g., `&r4` is an `ExtResource`). In the rest of the paper, the term *description base* will be used to denote a set of RDF statements. Although not illustrated in Figure 1, RDF also supports structured values called *containers* (i.e., bag, sequence) for grouping statements, as well as, higher-order statements (i.e., *reification*). Finally, both RDF graph schemas and descriptions can be serialized in XML using forests of XML trees.

We can observe that properties are *unordered* (e.g., the property *title* can be placed before or after the property *description*), *optional* (e.g., the property *file_size* is not used), can be *multi-valued* (e.g., we have two *description* properties), and they can be *inherited* (e.g., to subclasses of *ExtResource*), while resources

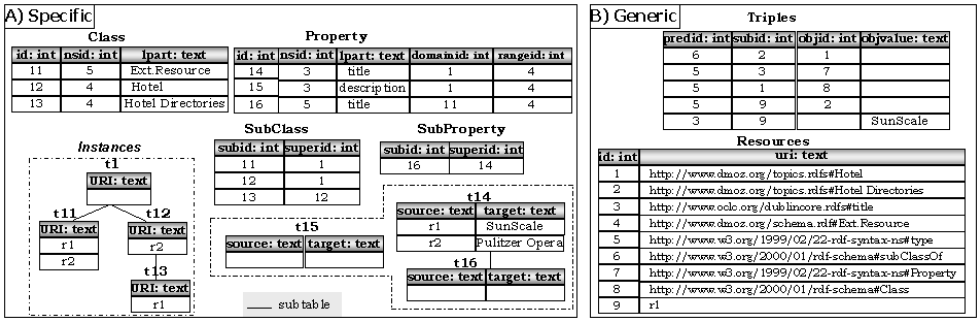

Figure 2: Two Relational Representations of RDF Description Bases

can be multiply classified (e.g., `&r4`). These modeling primitives provide all the flexibility we need to represent superimposed descriptions of Web resources for different syndication purposes (e.g., administration, retrieval), while preserving a conceptually unified view of the Portal Catalog (i.e., description base) through one or the union of all related schemas. Thus, community members can query resources described according to their preferred schema, while discover, in the sequel, how the same resources are also described using another (sub-)community schema. This is quite useful especially when different Community Web Portals need to exchange descriptions about their resources.

3 The RDF Storage System: Design and Performance

In order to load RDF metadata in a ORDBMS, we consider two basic representations: a *Specific Representation* (i.e., depending on the used RDF schemas) and a *Generic Representation* (i.e., for all RDF schemas). Both representations preserve the flexibility of RDF in refining schemas and/or enriching descriptions at any time. In the former representation (similar to the *attribute-based* approach for storing XML data [7]), called *SpecRepr*, the core RDF/S model is represented by four tables (see Figure 2-A), namely, `Class`, `Property`, `SubClass` and `SubProperty` which capture the class and property hierarchies defined in an RDF schema. Although not illustrated in Figure 2-A, we also consider a table `NameSpace` holding the namespaces of the RDF Schemas stored in the ORDBMS. The main goal of *SpecRepr* is the separation of the RDF schema from data information, as well, as the distinction between unary and binary relations holding the instances of classes and properties. More precisely, class tables store the URIs of resources while property tables store the URIs of the source and target nodes of the property. Indices (i.e., B-trees) are constructed on the attributes `URI`, `source` and `target` of the above tables, the attributes `lpart`, `nsid` and `id` of the tables `Class` and `Property` and on the attribute `subid` of the tables `SubClass` and `SubProperty`.

The latter representation (see Figure 2-B) has been proposed in [10, 9] and relies on a monolithic approach to represent RDF metadata under the form of triples. It is a generic representation (similar to the *edge-based* approach for storing XML data [7]), called *GenRepr*, in which both RDF schemas and resource descriptions are captured by two tables: `Resources` and `Triples`. The former table represents each resource (including schema classes and properties) by a unique *id* whereas the latter represents the statements made about the resources (including classes and properties) under the form of predicate-subject-object triples (captured by `predid`, `subid` and `objid` respectively). Note that in table `Triples` we distinguish between properties representing attributes (i.e., `objvalue` with literal values) and those relationships between resources (i.e., `objid` with URI object values). Indices are constructed for all table attributes.

Compared to *GenRepr*, our *SpecRepr* is flexible enough to allow the customization of the representation of RDF metadata in the underlying ORDBMS. This is important since no representation is good for all purposes and in most real-scale RDF applications variations of a basic representation are required to take into account the peculiarities of the employed schema classes and properties. Our aim here is to reduce the total number of created instance tables. This is justified by the fact that some commercial ORDBMSs (and not PostgreSQL) permit only a limited number of tables. Furthermore, numerous tables (e.g., the ODP catalog implies the creation of 252840 tables, i.e., one for each topic) have a significant overhead on the response time of all queries (i.e., to find and open a table, its attributes, etc.). One of the possible variations we have experimented for the ODP catalog is the representation of all class instances by a unique table `Instances` (see dashed rectangular in Figure 2-A). This table has two attributes, namely `uri` and `classid`, for keeping the uri's of the resources and the id's of the classes in which resources belong. The benefits of

this *SpecRepr* variation are illustrated in the sequel given that most ODP classes (i.e., topics) have few or no instances at all (more than 90% of the ODP topics contain less than 30 URIs). Another alternative variation to our basic *SpecRepr* could be the representation of properties with range a literal type, as attributes of the tables created for the domain of this property. Consequently, new attributes will be added to the created class tables. The tables created for properties with range a class will remain unchanged. The above representation is applicable to RDF schemas where attribute-properties are single-valued and they are not specialized.

For our performance study we used as testbed the RDF dump of the Open Directory Catalog (version of 16-01-2001). Experiments have been carried out on a Sun with two UltraSPARC-II 450MHz processors and 1 GB of main memory, using PostgreSQL (Version 7.0.2) with Unicode configuration. We have loaded 15 ODP hierarchies with a total number of 252825 topics contained in 51MB of RDF/XML files[2]. For these hierarchies, we have also loaded the corresponding descriptions of 1770781 resource URIs (672MB).

We have measured the database size required to load the ODP schema and resource descriptions in terms of triples in *SpecRepr* and *GenRepr*. In both representations the size of the DBMS scales linearly with the number of schema and data triples. The tests show that, in *SpecRepr*, each schema triple requires on average *0.086KB* versus *0.1582KB* in *GenRepr*. This is mainly attributed to the space saved in *SpecRepr* due to the `Namespace` table, as well as to the fact that for each class definition in *GenRepr* three tuples are stored: one in table `Resources` and two tuples in table `Triples` for the class itself and its unique superclass (i.e., 252825 extra tuples). The average time for loading a schema triple is about *0.0021 sec* and *0.0025 sec* respectively in the two representations. The time required to store a schema triple is larger in *GenRepr* because of extra tuples stored. When indices are constructed, the average storage volumes per schema triple become *0.1734KB* (*SpecRepr*) and *0.3062KB* (*GenRepr*) and the average loading times become *0.0025 sec* and *0.00317 sec* respectively. The average space required to store a data triple is *0.123KB* in both representations. This should not appear as a surprise since the extra space required in *SpecRepr* for storing the URIs of resources in the property tables compensates for the extra tuples (1770781) stored (one in table `Resources` and one in `Triples`) for each resource description in *GenRepr*. Note that we could obtain better storage volumes in *SpecRepr* by encoding the resource URIs as integers, as we did in *GenRepr*, but as we will see in the sequel this solution comes with extra loading and join costs (between the class and property tables) for the retrieval of the URIs. The tests also show that the average time for loading a data triple is about *0.0033 sec* and *0.0039 sec* respectively in the two representations. When indices are constructed, the average storage volumes per data triple become *0.2566KB* (*SpecRepr*) and *0.2706KB* (*GenRepr*) while the average loading time become *0.0043 sec* and *0.00457 sec* respectively. Despite the use of integer ids, the indices in *GenRepr* take up more space because of: a) the extra tuples stored b) the index constructed on the `predid` attribute for which there is no corresponding index in *SpecRepr*.

To summarize, after loading the entire ODP catalog, the size of tables in *GenRepr* is 545MB for `Triples` (5835756 tuples), 202MB for `Resources` (2022869 tuples) and the total size of indexes on these tables is 900MB. In *SpecRepr*, the size is 32MB for `Class` (252825 tuples), 8KB for `Property` (5 tuples), 11MB for `SubClass` (252825 tuples) and 0MB for `SubProperty`, and the total size of indexes on these tables is 44MB. The size of table `Instances` is 150MB (1770781 tuples) whereas that of the indices created on it is 140 MB.

Table 1 describes the RDF query templates that we used for our experiments, as well as their respective algebraic expressions in the two representations (capital letters abbreviate the table names of Figure 2). This benchmark illustrates the desired querying functionality for RDF description bases, namely: a) pure schema queries on class and property definitions (`Q1-Q4`); b) queries on resource descriptions using available schema knowledge (`Q5-Q9`); and c) schema queries for specific resource descriptions (`Q10`, `Q11`). These query templates exploit the advanced modeling features of RDF and they are useful in a variety of Community Web Portal applications. As a matter of fact, they depict the core functionality of our declarative query language for RDF, called *RQL*[3]. For example, using *RQL* we can express queries against the ODP catalog, such as: "Find the topics under the hierarchy Regional (i.e., schema query), containing resources for hotels in Paris whose title matches "*Opera*" (i.e., data query using schema information). In this context, the most frequently asked queries for Portals like ODP are: `Q2`,`Q3`,`Q5`,`Q8` and `Q9`. For the sake of accuracy, we carried out all benchmark queries several times: one initially to warm up the database buffers and then nine times to get the average execution time of a query. Table 2 illustrates the obtained execution time (in sec) for both representations in up to three different result cases per query. The main observation is that

[2] This is the volume of the pure ODP schema, produced when properties attributed to the classes are removed.
[3] See http://139.91.183.30:9090/RDF for further details.

Query	Description	Algebraic Expression in **GenRepr**	Algebraic Expression in **SpecRepr**
Q1	Find the range (or domain) of a property	$\sigma_{predid=9 \wedge subjid=\texttt{propid}}(T)$	$\sigma_{id=\texttt{propid}}(P)$
Q2	Find the direct subclasses of a class	$\sigma_{predid=6 \wedge objid=\texttt{clsid}}(T)$	$\sigma_{superid=\texttt{clsid}}(SC)$
Q3	Find the transitive subclasses of a class	$repeat \quad W_i \leftarrow (W_{i-1} \rtimes_{id=subjid}(\sigma_{predid=6}(T))) - W_{i-1}$ $until \quad W_i = W_{i-1}$	$repeat \quad W_i \leftarrow (W_{i-1} \rtimes_{id=superid} SC) - W_{i-1}$ $until \quad W_i = W_{i-1}$
Q4	Check if a class is a subclass of another class	$repeat \quad W_i \leftarrow (W_{i-1} \rtimes_{id=objid}(\sigma_{predid=6}(T))) - W_{i-1}$ $until \quad W_i = W_{i-1} \vee \texttt{clsid} \in W_i$	$repeat \quad W_i \leftarrow (W_{i-1} \rtimes_{id=subid} SC) - W_{i-1}$ $until \quad W_i = W_{i-1} \vee \texttt{clsid} \in W_i$
Q5	Find the direct extent of a class (or property)	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \rtimes_{subjid=id} R$	$\sigma_{id=\texttt{clsid}}(I)$
Q6	Find the transitive extent of a class (or property)	$\cup_{clsid \in \texttt{Q3}}((\sigma_{predid=5 \wedge objid=clsid}(T)) \rtimes_{subjid=id} R)$	$\cup_{clsid \in \texttt{Q3}}(\sigma_{id=clsid}(I))$
Q7	Find if a resource is an instance of a class	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \bowtie_{subjid=id} (\sigma_{URI=\texttt{r}}(R))$	$\sigma_{URI=\texttt{r} \wedge id=\texttt{clsid}}(I)$
Q8	Find the resources having a property with a specific (or range of) value(s)	$(\sigma_{predid=\texttt{propid} \wedge objvalue=\texttt{val}}(T)) \bowtie_{subjid=id} R$	$\sigma_{target=\texttt{val}}(t_{\texttt{propid}})$
Q9	Find the instances of a class that have a given property	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \rtimes_{subjid=subjid}(\sigma_{predid=\texttt{propid}}(T)) \rtimes_{subjid=id}(R)$	$(\sigma_{id=\texttt{clsid}}(I)) \rtimes_{source=URI} (t_{\texttt{propid}})$
Q10	Find the properties of a resource and their values	$(\sigma_{URI=\texttt{r}}(R)) \bowtie_{id=subjid} (\sigma_{predid \neq 5}(T)) \bowtie_{predid=id} (R)$	$\cup_{propid \in P}(\sigma_{source=\texttt{r}}(t_{\texttt{propid}}))$
Q11	Find the classes under which a resource is classified	$(\sigma_{URI=\texttt{r}}(R)) \rtimes_{id=subjid} \sigma_{predid=6}(T)$	$\sigma_{URI=\texttt{r}}(I)$

Table 1: Benchmark Query Templates for RDF Description Bases

SpecRepr outperforms *GenRepr* for all types of queries considered. The deviation in performance is more apparent in the cases where self-join computations on the large `Triples` table are required.

GenRepr and *SpecRepr* exhibit comparable performance in queries `Q1, Q2, Q5, Q7, Q10` and `Q11`, with *SpecRepr* outperforming *GenRepr* by a factor of up to 3.73 approximately. In `Q1` one tuple is selected from both table `Triples` (selectivity 1,7e-5%) and `Property` (selectivity 20%) using index and sequential scans respectively. In `Q2`, we can see that the time required to filter a table in both representations depends on the number of tuples in the query results: we have experimented with classes having 1, 30, 314 subclasses which represent in *GenRepr* (*SpecRepr*) selectivities of 1.7e-5% (3.955e-4%), 5.14e-4% (1,19e-2%) and 5.38e-3% (0.124%) for table `Triples` (`SubClass`) in the three cases respectively. The (semi-)joins involved in the evaluation of queries `Q5`, `Q7` and `Q11` incur an additional cost for *GenRepr*, whereas in `Q10` the join cost (for *GenRepr*) is comparable to the cost of evaluating set union (for *SpecRepr*). Queries `Q3`, `Q4` and `Q6` involve a transitive closure computation (using a variation of the *δ-wavefront* algorithm [11]) over the subclass hierarchy. *SpecRepr* outperforms *GenRepr* by a factor of up to 2.8. In `Q3` and `Q6`, we use the same three classes having 3, 30 and 3879 subclasses and a total of 2, 20 and 9049 instances respectively. The execution times in these three cases depend on the sizes of intermediate results (i.e., the costs of joins involving the tables `Triples` or `SubClass`) as well as, the number of iteration steps of the algorithm (i.e., the length of the longest path from the given class to its leaves, called *depth*). In `Q4`, for the same root class, we have checked for subclasses residing at depth 3, 5 and 7 respectively. The difference in the obtained times between `Q3, Q6` and `Q4` is due to the different evaluation method used: "top-down" for the former (i.e., from the sub-tree root to the leaves) and "bottom-up" for the latter.

In the case of queries `Q8` and `Q9` *SpecRepr* exhibits a much better performance than *GenRepr*. *GenRepr* reaches its limits when table `Triples` needs to be self-joined whereas in *SpecRepr*, a join between two small tables suffices to be evaluated. Specifically, *SpecRepr* outperforms *GenRepr* by a factor ranging from 1753 up

Query	Generic			Specific		
	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3
Q1	0.0015			0.0012		
Q2	0.0017	0.0028	0.02	0.0012	0.0022	0.0124
Q3	0.0460	0.082	344.91	0.0463	0.0612	341.98
Q4	0.033	0.0415	0.0662	0.0333	0.0415	0.0662
Q5	0.0043	0.008	0.04	0.0015	0.0028	0.027
Q6	0.0573	0.315	627.43	0.0508	0.1118	482.45
Q7	0.0034	0.0034	0.0034	0.0016	0.0016	0.00174
Q8	124.20	365.73	675.42	0.0013	0.0069	0.0466
Q9	110.58	117.68	185.7	0.031	0.0338	0.1059
Q10	0.0072	0.0072	0.0072	0.0071	0.0071	0.0076
Q11	0.0035	0.0043	0.0056	0.0013	0.0015	0.0015

Table 2: Execution Time of RDF Benchmark Queries

to 95538. Note that *GenRep* will suffer similar performance limitations in the evaluation of queries involving complex path expressions which will essentially result in a number of self-joins of table `Triples`. Query `Q8` has been tested for value ranges returning 1, 10 and 40 resources respectively. In *SpecRepr*, its evaluation involves index scans on the property table, whereas in *GenRepr* different evaluation plans are executed in each case. `Q9` has been tested for three properties with 6292, 52029 and 1770584 instances respectively. To summarize, *SpecRepr* outperforms *GenRepr*, which pays a severe performance penalty for maintaining large tables. We argue that the performance of *SpecRepr* can be further improved by employing an appropriate encoding system (e.g., Dewey, postfix, prefix, etc.) that preserves the taxonomic relationships of schema labels. In this way, checking subclass relationships can be done in constant time. We believe that this approach will prove to be quite useful, not only for RDF, but for tree-structured data, such as XML [2].

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *12th Symposium on Discrete Algorithms*, 2001.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] G. Beged-Dov, D. Brickley, R. Dornfest, I. Davis, L. Dodds, J. Eisenzopf, D. Galbraith, R. Guha, E. Miller, and E. van der Vlist. RSS 1.0 Specification Protocol. (http://purl.org/rss/1.0), August 2000.

[5] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Recommendation. Technical Report CR-rdf-schema-20000327, W3C, (http://www.w3.org/TR/rdf-schema), March 27, 2000.

[6] C. Finkelstein and P. Aiken. *Building Corporate Portals using XML*. McGraw-Hill, 1999.

[7] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical Report 3680, INRIA Rocquencourt, France, May 1999. (http://www-caravel.inria.fr/-dataFiles/GFSS00.ps).

[8] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, W3C, February 1999. (http://www.w3.org/TR/REC-rdf-syntax).

[9] J. Liljegren. Description of an rdf database implementation. Available at http://WWW-DB.stanford.edu/~melnik/rdf/-db-jonas.html.

[10] S. Melnik. Storing rdf in a relational database. Available at http://WWW-DB.stanford.edu/~melnik/rdf/db.html.

[11] G. Qadah, L. Henschen, and J. Kim. Efficient Algorithms for the Instantiated Transitive Closure Queries. *IEEE Transactions on Software Engineering*, 17(3):296–309, 1991.

[12] Albrecht Schmidt, Martin L. Kersten, Menzo Windhouwer, and Florian Waas. Efficient relational storage and retrieval of xml documents. In *WebDB'00*, pages 47–52, 2000.

[13] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D.J. DeWitt, and J.F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *VLDB'99*, pages 302–314, Edinburgh, Scotland, September 1999.

[14] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. Technical report, Universiy of Wisconsin, 2000.

[15] S. Weibel, J. Miller, and R. Daniel. Dublin Core. In *OCLC/NCSA metadata workshop report*, 1995.

[16] Web Service Description Language. (http://www-106.ibm.com/developerworks/library/ws-rdf), 2000.