Accessing Relational Data with RDF Queries and Assertions

Dmitry Borodaenko

angdraug@debian.org

Abstract. This paper presents a hybrid RDF storage model that combines relational data with arbitrary RDF meta-data, as implemented in the RDF storage layer of the Samizdat open publishing and collaboration engine, and explains the supporting algorithms for online translation of RDF queries and conditional assertions into their relational equivalents. Proposed model allows to supplement legacy databases with RDF meta-data without sacrificing the benefits of RDBMS technology.

1 Introduction

The survey of free software / open source RDF storage systems performed by SWAD-Europe[2] has found that the most wide-spread approach to RDF storage relies on relational databases. As seen from the companion report on mapping Semantic Web data with RDBMSes[3], traditional relational representation of RDF is a triple store, usually evolving around a central statement table with {subject, predicate, object} triples as its rows and one or more tables storing resource URIrefs, namespaces, and other supplementary data.

While such triple store approach serves well to satisfy the open world assumption of RDF, by abandoning existing relational data models it fails to take full advantage of the RDBMS technology. According to [2], existing RDF storage tools are still immature; in the same time, although modern triple stores claim to scale to millions of triples, ICS-FORTH research[1] shows that schema-specific storage model yields better results with regards to performance and scalability on large volumes of data.

These concerns are addressed from different angles by RSSDB[12], Federate[11], and D2R[6] packages. RSSDB splits the single triples table into a schema-specific set of property tables. In this way, it walks away from relational data model, but maintains performance benefits due to better indexing. Federate takes the most conservative approach and allows to query a relational database with a restricted application-specific RDF schema. Conversely, D2R is intended for batch export of data from RDBMS to RDF and assumes that subsequent operation will involve only RDF.

The hybrid RDF storage model presented in this paper attacks this problem from yet another angle, which can be described as a combination of Federate's relational-to-RDF mapping and a traditional triple store. While having the advantage of being designed from the ground up with the RDF model in mind,

Samizdat RDF layer[7] deviated from the common RDF storage practice in order to use both relational and triple data models and get the best of both worlds. Hybrid storage model was designed, and algorithms were implemented that allow to access the data in the hybrid triple-relational model with RDF queries and conditional assertions in an extended variant of the Squish[13] query language.[1] This paper describes the proposed model and its implementation in the Samizdat engine.

2 Relational Database Schema

All content in a Samizdat site is represented internally as RDF. Canonic URIref for any Samizdat resource is `http://<site-url>/<resource-id>`, where `<site-url>` is a base URL of the site and `<resource-id>` is a unique (within a single site) numeric identifier of the resource.

Root of SQL representation of RDF resources is `Resource` table with `id` primary key field storing `<resource-id>`, and `label` text field representing resource label. Semantics of label values are different for literals, references to external resources, and internal resources of the site.

Literal value (including typed literals) is stored directly in the `label` field and marked with `literal` boolean field.

External resource label contains the resource URIref and is marked with `uriref` boolean field.

Internal resource is mapped into a row in an *internal resource table* with name corresponding to the resource class name stored in the `label` field, primary key `id` field referencing back to the `Resource` table, and other fields holding values of *internal properties* for this resource class, represented as literals or references to other resources stored in the `Resource` table. Primary key reference to `Resource.id` is enforced by PostgreSQL stored procedures.

To determine what information about a resource can be stored in and extracted from class-specific tables, RDF storage layer consults site-specific mapping

$$M(p) = \{\langle t_{p1},\ f_{p1}\rangle,\ \ldots\}\ , \tag{1}$$

which stores a list of possible pairs of SQL table name t and field name f for each internal property name p. Mapping M is read at runtime from external YAML[4] file of the following form:

```
---
ns:
  s: 'http://www.nongnu.org/samizdat/rdf/schema#'
  focus: 'http://www.nongnu.org/samizdat/rdf/focus#'
```

[1] The decision to use Squish over more expressive languages like RDQL[10] and Notation3[5] was made due to its intuitive syntax, which was found more suitable for the Samizdat's query composer GUI intended for end users of an open-publishing system.

```
  items: 'http://www.nongnu.org/samizdat/rdf/items#'
  rdf: 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  dc: 'http://purl.org/dc/elements/1.1/'

map:
  'dc::date': {Resource: published_date}
  's::id': {Resource: id}

  'rdf::subject': {Statement: subject}
  'rdf::predicate': {Statement: predicate}
  'rdf::object': {Statement: object}

  's::rating': {Statement: rating}

  . . .
```

External properties, i.e. properties that are not covered by M, are represented by {`subject`, `predicate`, `object`} triples in the `Statement` table. Every such triple is treated as a reified statement in RDF semantics and is assigned a `<resource-id>` and a record in the `Resource` table.

`Resource` and `Statement` are also internal resource tables, and, as such, have some of their fields mapped by M. In particular, `subject`, `predicate`, and `object` fields of the `Statement` table are mapped to the corresponding properties from the RDF reification vocabulary, and `Resource.id` is mapped to `samizdat:id` property from Samizdat namespace.

Excerpt from default Samizdat database schema with mapped field names replaced by predicate QNames is visualized on Fig. 1. In addition to `Resource` and `Statement` tables described above, it shows the `Message` table representing one of internal resource classes. Note how `dc:date` property is made available to all resource classes, and how reified statements are allowed to have optional `samizdat:rating` property.

3 Query Pattern Translation

3.1 Prerequisites

Pattern translation algorithm operates on the pattern section of a Squish query. Query pattern Ψ is represented as a list of *pattern clauses*

$$\psi_i = \langle p_i,\ s_i,\ o_i \rangle\ , \tag{2}$$

where i is the position of a clause, p_i is the predicate URIref, s_i is the subject node and may be URIref or blank node, o_i is the object node and may be URIref, blank node, or literal.

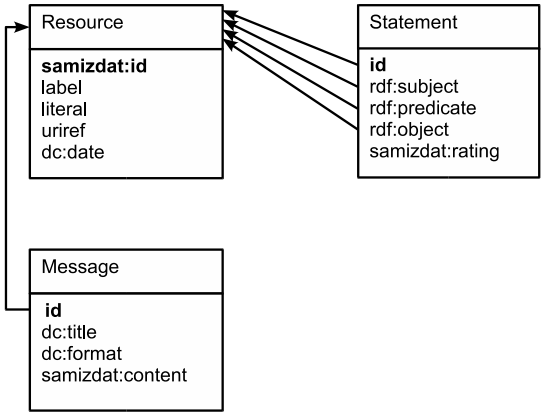

Fig. 1. Excerpt from Samizdat database schema

3.2 Predicate Mapping

For each position i, predicate URIref p_i is looked up in the map of internal resource properties M. All possible mappings are recorded for all clauses in a list C:

$$c_i = \{\langle t_{i1},\ f_{i1}\rangle,\ \langle t_{i2},\ f_{i2}\rangle,\ \ldots\}\ , \tag{3}$$

where t_{ij} is the table name (same for subject s_i and object o_i) and f_{ij} is the field name (meaningful for object only, since subject is always mapped to the `id` primary key). In the same iteration, all subject and object positions of nodes are recorded in the reverse positional mapping

$$R(n) = \{\langle i_1,\ m_1\rangle,\ \langle i_2,\ m_2\rangle,\ \ldots\}\ , \tag{4}$$

where m shows whether node n appears as subject or as object in the clause i.

Each ambiguous property mapping is compared with mappings for other occurrences of the same subject and object nodes in the pattern graph; anytime non-empty intersection of mappings for the same node is found, both subject and object mappings for the ambiguous property are refined to such intersection.

3.3 Relation Aliases and Join Conditions

Relation alias a_i is determined for each clause mapping c_i, such that for all subject occurrences of the subject s_i that were mapped to the same table t_i, alias is the same, and for all positions with differing table mapping or subject node, alias is different.

For all nodes n that are mapped to more than one $\langle a_i,\ f_i\rangle$ pair in different positions, join conditions are generated. Additionally, for each external resource, `Resource` table is joined by URIref, and for each existential blank node that isn't already bound by join, `NOT NULL` condition is generated. Resulting join conditions set J is used to generate the `WHERE` section of the target SQL query.

3.4 Example

Following Squish query selects all messages with rating above 1:

```
SELECT ?msg, ?title, ?name, ?date, ?rating
WHERE (dc::title ?msg ?title)
      (dc::creator ?msg ?author)
      (s::fullName ?author ?name)
      (dc::date ?msg ?date)
      (rdf::subject ?stmt ?msg)
      (rdf::predicate ?stmt dc::relation)
      (rdf::object ?stmt focus::Quality)
      (s::rating ?stmt ?rating)
LITERAL ?rating >= 1
ORDER BY ?rating
USING rdf FOR http://www.w3.org/1999/02/22-rdf-syntax-ns#
      dc FOR http://purl.org/dc/elements/1.1/
      s FOR http://www.nongnu.org/samizdat/rdf/schema#
      focus FOR http://www.nongnu.org/samizdat/rdf/focus#
```

Mappings produced by translation of this query are summarized in the Table 1.

Table 1. Query Translation Mappings

i	t_i	f_i	a_i
1	Message	title	b
2	Message	creator	b
3	Member	full_name	d
4	Resource	published_date	c
5	Statement	subject	a
6	Statement	predicate	a
7	Statement	object	a
8	Statement	rating	a

As a result of translation, following SQL query will be generated:

```
SELECT b.id, b.title, d.full_name, c.published_date, a.rating
FROM Statement a, Message b, Resource c, Member d,
     Resource e, Resource f
WHERE a.id IS NOT NULL
  AND a.object = e.id AND e.literal = false
      AND e.uriref = true AND e.label = 'focus::Quality'
  AND a.predicate = f.id AND f.literal = false
```

```
      AND f.uriref = true AND f.label = 'dc::relation'
  AND a.rating IS NOT NULL
  AND b.creator = d.id
  AND b.id = a.subject
  AND b.id = c.id
  AND b.title IS NOT NULL
  AND c.published_date IS NOT NULL
  AND d.full_name IS NOT NULL
  AND (a.rating >= 1)
ORDER BY a.rating
```

3.5 Limitations

In RDF model theory[9], a resource may belong to more than one class. In Samizdat RDF storage model, resource class specified in `Resource.label` is treated as the primary class: it is not possible to have some of the internal properties of a resource mapped to one table and some other internal properties mapped to the other. The only exception to this is, obviously, the `Resource` table, which is shared by all resource classes.

Predicates with cardinality greater than 1 cannot be mapped to internal resource tables, and should be recorded as reified statements instead.

RDF properties are allowed to be mapped to more than one internal resource table, and queries on such ambiguous properties are intended to select all classes of resources that match this property in conjunction with the rest of the query.

The algorithm described above assumes that other pattern clauses refine such ambiguous property mapping to one internal resource table. Queries that fail this assumption will be translated incorrectly by the current implementation: only the resource class from the first remaining mapping will be matched. This should be taken into account in site-specific resource maps: ambiguous properties should be avoided where possible, and their mappings should go in order of resource class probability descension.

It is possible to solve this problem, but any precise solution will add significant complexity to the resulting query. Solutions that would not adversely affect performance are still being sought. So far, it is recommended not to specify more than one mapping per internal property.

4 Conditional Assertion

4.1 Prerequisites

Conditional assertion statement in Samizdat Squish is recorded using the same syntax as RDF query, with the `SELECT` section containing variables list replaced by `INSERT` section with a list of "don't-bind" variables and `UPDATE` section containing assignments of values to query variables:

```
[ INSERT node [, ...] ]
[ UPDATE node = value [, ...] ]
WHERE (predicate subject object) [...]
[ USING prefix FOR namespace [...] ]
```

Initially, pattern clauses in assertion are translated using the same procedure as for a query. Pattern Ψ, clause mapping C, reverse positional mapping R, alias list A, and join conditions set J are generated as described in the previous section.

After that, database update is performed in two stages described below. Both stages are executed within a single transaction, rolling back intermediate inserts and updates in case assertion fails.

4.2 Resource Values

On this stage value mapping $V(n)$ is defined for each node n, and necessary resource insertions are performed:

1. If n is an internal resource, $V(n)$ is its `id`. If there is no resource with such `id` in the database, error is raised.
2. If n is a literal, $V(n)$ is the literal value.
3. If n is a blank node and only appears in object position, it is assigned a value from the `UPDATE` section of the assertion.
4. If n is a blank node and appears in subject position, it is either looked up in the database or inserted as a new resource. If no resource in the database matches n (to check that, subgraph of Ψ including all pattern nodes and predicates reachable from n is generated and matched against the database), or if n appears in the `INSERT` section of the assertion, new resource is created and its `id` is assigned to $V(n)$. If matching resource is found, $V(n)$ becomes equal to its `id`.
5. If n is an external URIref, it is looked up in the `Resource` table. As with subject blank nodes, $V(n)$ is the `id` of a matching or new resource.

All nodes that were inserted during this stage are recorded in the set of new nodes N.

4.3 Data Assignment

For all aliases from A except additional aliases that are defined for external URIref nodes (which don't have to be looked up since their `id`s are recorded in V during the previous stage), reverse positional mapping

$$R_{\mathrm{A}}(a) = \{i_1,\ i_2,\ \ldots\} \tag{5}$$

is defined. Key node K is defined as the subject node s_{i_1} from clause ψ_{i_1}, and aliased table t is defined as the table name t_{i_1} from clause mapping c_{i_1}.

For each position k from $R_{\mathrm{A}}(a)$, a pair $\langle f_k, V(o_k)\rangle$, where f_k is the field name from c_k, and o_k the object node from ψ_k, is added to the data assignment list $D(K)$ if node o_k occurs in new node list N or in `UPDATE` section of the assertion statement.

If key node K occurs in N, new row is inserted into the table t. If K is not in N, but $D(K)$ is not empty, SQL update statement is generated for the row of t with `id` equal to $V(K)$. In both cases, assignments are generated from the data assignment list $D(K)$.

The above procedure is repeated for each alias a included in R_{A}.

4.4 Iterative assertions

If the assertion pattern matches more than once in the site knowledge base, the algorithm defined in this section will nevertheless run the appropriate insertions and updates only once. For iterative update of all occurences of pattern, assertion has to be programmatically wrapped inside an appropriate RDF query.

5 Implementation Details

Samizdat engine[8] is written in Ruby programming language and uses PostgreSQL database for storage and an assortment of Ruby libraries for database access (DBI), configuration and RDF mapping (YAML), l10n (GetText), and Pingback protocol (XML-RPC). It is running on a variety of platforms ranging from Debian GNU/Linux to Windows 98/Cygwin. Samizdat is free software and is available under GNU General Public License, version 2 or later.

Samizdat project development started in December 2002, first public release was announced in June 2003. As of the second beta version 0.5.1, released in March 2004, Samizdat provided basic set of open publishing functionality, including registering site members, publishing and replying to messages, uploading multimedia messages, voting on relation of site focuses to resources, creating and managing new focuses, hand-editing or using GUI for constructing and publishing Squish queries that can be used to search and filter site resources.

6 Conclusions

Wide adoption of the Semantic Web requires interoperability between relational databases and RDF applications. Existing RDF stores treat relational data as legacy and require that it is recorded in triples before being processed, with the exception of the Federate system that provides limited direct access to relational data via application-specific RDF schema.

The Samizdat RDF storage layer provides an intermediate solution for this problem by combining relational databases with arbitrary RDF meta-data. The described approach allows to take advantage of RDBMS transactions, replication, performance optimizations, etc., in Semantic Web applications, and reduces the costs of migration from relational data model to RDF.

As can be seen from corresponding sections of this paper, current implementation of the proposed approach has several limitations. These limitations are not caused by limitations in the approach itself, but rather, reflect the pragmatic decision to only implement the functionality that is used by Samizdat engine. As more advanced collaboration features such as message versioning and aggregation are added to Samizdat, some of the limitations of its RDF storage layer will be removed.

References

1. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis D., Tolle, K.: The RDFSuite: Managing Voluminous RDF Description Bases, Technical report, ICS-FORTH, Heraklion, Greece, 2000.
   http://139.91.183.30:9090/RDF/publications/semweb2001.html
2. Beckett, Dave: Semantic Web Scalability and Storage: Survey of Free Software / Open Source RDF storage systems, SWAD-Europe Deliverable 10.1
   http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report
3. Beckett, D., Grant, J.: Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes, SWAD-Europe Deliverable 10.2
   http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report
4. Ben-Kiki, O., Evans, C., Ingerson, B.: YAML Ain't Markup Language (YAML) 1.0. Working Draft 2004-JAN-29.
   http://www.yaml.org/spec/
5. Berners-Lee, Tim: Notation3 — Ideas about Web architecture
   http://www.w3.org/DesignIssues/Notation3
6. Bizer, Chris: D2R MAP — Database to RDF Mapping Language and Processor
   http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rmap/D2Rmap.htm
7. Borodaenko, Dmitry: Samizdat RDF Storage, December 2002
   http://savannah.nongnu.org/cgi-bin/viewcvs/samizdat/samizdat/doc/rdf-storage.txt
8. Borodaenko, Dmitry: Samizdat RDF Implementation Report, September 2003
   http://lists.w3.org/Archives/Public/www-rdf-interest/2003Sep/0043.html
9. Hayes, Patrick: RDF Semantics. W3C, February 2004
   http://www.w3.org/TR/rdf-mt
10. Jena Semantic Web Framework: RDQL Grammar
    http://jena.sf.net/RDQL/rdql_grammar.html
11. Prud'hommeaux, Eric: RDF Access to Relational Databases
    http://www.w3.org/2003/01/21-RDF-RDB-access/
12. RSSDB — RDF Schema Specific DataBase (RSSDB), ICS-FORTH, 2002
    http://139.91.183.30:9090/RDF/RSSDB/
13. Libby Miller, Andy Seaborne, Alberto Reggiori: Three Implementations of SquishQL, a Simple RDF Query Language. 1st International Semantic Web Conference (ISWC2002), June 9-12, 2002. Sardinia, Italy.
    http://ilrt.org/discovery/2001/02/squish/