1. A relational model for VOResource

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

1.1 Why?
1.2 The tables
1.3 Some queries
1.4 The haswords ADQL function

2. Why?

The current interface for searchable registries relies on ADQL 1.0, which has never made it to REC, and the implementations are of inconsistent quality.

We now have TAP and a REC-status ADQL. ObsCore shows how to define tables in TAP servers.

Let’s define RegTAP!

Thoughts have been collected on the RestfulRegistryInterfaceReq wiki page.

3. The Tables: How?

Paul Harrison translated the VOResource schema and much of the vicinity (VODataService, SimpleDALRegExt...) to a relational schema using

- JAXB to go from XML to Java objects and then
- JPA (plus a lot of tweaking) to turn those into relational tables.

I used this as a starting point to “hand-optimize” tables. Additionally, I created utypes basically as XPaths into VOResource trees.

Here they are (boredom alert – 14 table definitions ahead. Bear with me, we need to discuss this).

4. resource 1/3

<table>
<thead>
<tr>
<th>ivoid</th>
<th>vor:Resource.identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_type</td>
<td>vor:Resource.type</td>
</tr>
<tr>
<td>created</td>
<td>vor:Resource.created</td>
</tr>
<tr>
<td>shortname</td>
<td>vor:Resource.shortName</td>
</tr>
<tr>
<td>status</td>
<td>vor:Resource.status</td>
</tr>
<tr>
<td>title</td>
<td>vor:Resource.title</td>
</tr>
<tr>
<td>updated</td>
<td>vor:Resource.updated</td>
</tr>
<tr>
<td>contentlevel</td>
<td>vor:Resource.content.contentLevel</td>
</tr>
<tr>
<td>new_description</td>
<td>vor:Resource.content.description</td>
</tr>
<tr>
<td>referenceURL</td>
<td>vor:Resource.content.referenceURL</td>
</tr>
<tr>
<td>contact_address</td>
<td>vor:Resource.curation.contact.address</td>
</tr>
</tbody>
</table>

Observations: All tables have a column ivoid; most joins will use it. It’s also this table’s primary key.

Most columns here translate easily from VOResource to the relational model. The utypes lead you to the schema equivalent.

5. resource 2/3

| contact_email | vor:Resource.curation.contact.email |
| contact_phone | vor:Resource.curation.contact.telephone |
| contact_name | vor:Resource.curation.contact.name.ivo-id |
| contact_name | vor:Resource.curation.contact.name |
| subject       | vor:Resource.content.subject |
| content_type  | vor:Resource.content.type |
| source_format | vor:Resource.content.sourceformat |
| source_value  | vor:Resource.content.source |
| version       | vor:Resource.curation.version |

Here’s a simplification: Only one contact is allowed (our reaper discards all but the last one). Rationale: Multiple contacts are bewildering anyway, and keeping that isn’t worth the cost of an extra table.

subject is supposed to be a comma-separated list. Having an extra table here might be nice (though – it wouldn’t, e.g., provide an easy way to figure out a complete vocabulary. It would discard subject from – ), so I’d be easily convinced.

There’s the compound source element flattened here. It is understood that all corresponding fields get NULLed when the parent element is missing.
The main evil things here are covered_region and stc_profile. In my current implementation, the first is completely missing, where the second simply is an STC-S representation of whatever STC-X came with the RR. I don’t really see how to represent what subset of STC here. Temporal coverage? Spectral coverage? How about spatial unions and intersections? harvested_from is housekeeping and won’t be part of the standard.

This table references capabilities and should contain utype-value pairs for stuff from registry extensions. Example:
```
gavo=# select top 3 detail_utype, detail_value from rr.capability_detail;
```

<table>
<thead>
<tr>
<th>detail_utype</th>
<th>detail_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource.capability.imageServiceType</td>
<td>CutoutResource.capability.maxFileSize</td>
</tr>
<tr>
<td>Resource.capability.maxRecords</td>
<td>500</td>
</tr>
</tbody>
</table>

Enumerations are still possible, but no parent/child relations. That would be a problem if we wanted to represent all of TAPRegExt in the database.

This subsumes the attributes of the various VOResource interfaces.

We could fold this table into interface if we allowed only one access URL per interface. This would make many queries much nicer. Is there anyone out there who’d be hurt by that?
11. intf_param

This references interface.

12. tableschema

This is pretty much like capability, except it is referenced by the rsctable table and thus collects tables.

13. rsctable

The primary key here is (ivoid, table_index), i.e., we table indices must be unique within a resource rather than a schema. This helps when referencing this from tablecolumn.

14. tablecolumn

This table has no primary key since it is not referenced from anywhere. The datatype is a bit of a troublemaker here. VODataService allows these to be either TAPDataType or VOTableDataType, and this is currently not reflected here.

On the other hand – do we actually want delim here? extended whatever?

15. relationship

16. creator

We need this since a resource can have multiple creators. In some bright future, this may be used to save people the hassle of having to parse author lists, which would be Great Progress.

Of course, everyone has author lists in the creator field right now, and it’s unclear whether many authors will have ivoids. I could thus be swayed to strike this table.
17. resource, capabilityvalidation

<table>
<thead>
<tr>
<th>ivoid</th>
<th>N/A</th>
<th>validatedby</th>
<th>vor:Resource.validationLevel.validatedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>vor:Resource.validationLevel</td>
<td>cap</td>
<td>cap.index</td>
</tr>
<tr>
<td>validatedby</td>
<td>vor:Resource.capability.validationLevel.validatedBy</td>
<td>level</td>
<td>vor:Resource.capability.validationLevel</td>
</tr>
</tbody>
</table>

Merge those? cap.index is NULL would work out nicely as a marker for resource-level validation, but of course you couldn’t have a foreign key then...

18. rdate

<table>
<thead>
<tr>
<th>ivoid</th>
<th>N/A</th>
<th>date_value</th>
<th>vor:Resource.curation.date.value</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>vor:Resource.curation.date.role</td>
<td>cap</td>
<td>cap.index</td>
</tr>
</tbody>
</table>

You’ve done it! These are all tables I’m proposing for a searchable registry. That’s 14 in all, with maybe 2 or 3 we could still save.

19. Digression: More tables

In implementation, I needed some additional tables:

- registries – registries I’m harvesting with dates of last full and incremental harvests
- imported – a table keeping track of which harvested files have already been ingested
- authorities – what registries claim authority over which authorities? I’m ignoring all records from registries that aren’t authority for the record’s ivoid, except when there’s no managing registry at all; this table lets me do this.
- oaisets – for the OAI-PMH interface, I keep track of OAI sets declared (that’s not worth it)
- oairecs – preformatted OAI-PMH records by ivoid for the OAI-PMH interface

20. Use cases I

On the wiki page, some use cases were presented. Those that can be answered via VOResource can be answered using this schema (see http://docs.g-vo.org/rr-usecases.rstx).

Some examples:

“Find all SIA services that provides infrared images”

```sql
SELECT ivoid, url
FROM rr.capability
NATURAL JOIN rr.resource
NATURAL JOIN rr.accessurl
WHERE standard_id='ivo://ivoa.net/std/SIA'
AND waveband LIKE '%Infrared%'
```

21. Use cases II

“Find all searchable catalogs that provide a column containing redshift”

```sql
SELECT ivoid, url
FROM rr.capability
NATURAL JOIN rr.tablecolumn
WHERE standard_id='ivo://ivoa.net/std/ConeSearch'
AND ucd='src.redshift'
```

“Find all SIA services that provides infrared images”

```sql
SELECT ivoid, url
FROM rr.capability
NATURAL JOIN rr.resource
NATURAL JOIN rr.accessurl
WHERE standard_id='ivo://ivoa.net/std/SIA'
AND 1=ivo_hasword('infrared', waveband)
```

22. The hasword function

We should mandate a function (say)

```python
ivo_hasword(needle, haystack) -> integer
```

returning 1 if needle is contained in haystack “as a word” and in any capitalization. This should be more or less like Postgres’ information retrieval functions. We want this for matching descriptions, authors, etc.

23. Open questions

- Should the table names be plural forms?
- Can we save the accessurl table? The creator table? The rdate table? Either one of the validation tables?
- Erm – I left out securityMethod. Just add it as a text field in interface
- Where do we go from here?