1. TAP evolution: examples and plan

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

- Why examples?
- Requirements for examples
- A microformat.
- Can we have a plan?

2. Why examples?

Informal user studies as well as code review invariably indicate:

- Users love examples
- chances they'll use a service are much better if they can start with something that works, even if it's quite different from what they need.

3. Requirements for examples

- Examples may range from almost uncommented one-liners to tutorial chapters
- Examples must be easily discoverable
- Protocol clients should at least be aware of examples and be able to fill example queries into their forms
- Example authors should have rich markup (i.e., at least links, paragraphs, verbatim text, and emphasis should be supported)
- Protocol clients wishing to display example documentation inline should have a fair chance to do so
- Linking examples to specific tables would be nice

4. Microformats

A microformat basically is HTML with light additional markup to enable extraction of structured information; there are microformats for calendar entries, addresses, etc. See also microformats.org[1].

Advantages:

- Rich markup
- Lots of renderers
- Everyone knows how to write HTML

Disadvantages:

- HTML is a big, moving target
- General HTML is a pain to parse

[1] `http://microformats.org`

5. Microformat: Container Issues

We're based on well-formed XHTML, but any XHTML is allowed. Elements are identified by CSS classes (i.e., words within a `class` attribute value).

Each example is an element with a class of `ivo_tap_example`. It must have an `id` attribute.

This would usually be a `div`.

Within each example, there must be a unique element with simple text content and class `ivo_tap_examplename` suitable for display in labels or combo boxes.

This would usually be a `h2`

6. Microformat: Query, Table

To communicate the query, have an element with simple content and a CSS class `ivo_tap_examplequery`.

That will ususally be a `pre`.

You can associate the query with one or more elements with simple content and a CSS class `ivo_tap_exampletable`.

You could use `abbrev` or `a` elements for that.

7. Example Example

```
<div class="ivo_tap_example" id="katkatbib">
  <h1 class="ivo_tap_examplename">Katkat Bibliography</h1>

  <p>To search for title (or other) words in the source field of <abbr
  class="ivo_tap_exampletable">katkat.katkat</abbr> use the
  <tt>gavo_hasword</tt> locally defined function. This basically works
  a bit like you'd expect from search engines: case-insensitive, and
  oblivious to any context.</p>

  <pre class="ivo_tap_examplequery">
  SELECT *
  FROM katkat.katkat
  WHERE gavo_hasword('variable', source)
    AND minEpoch<1900
  </pre>
</div>
```

8. We need a Plan!

SQL queries are expressions of relational algebra that databases "simplify" before execution.

Knowing what simplification a server chose is paramount to analyze why queries are slow.

So: TAP needs a way to communicate query plans.

But: Query plans are hard to read, every DMBS has their own way of writing those, etc.

Proposal: Let's have `.../plan` within each job resource (alongside `.../phase` and siblings).

Should `plan` return plain text or XML?

9. Plan Plain Text Example

It's not clear that any client would ever want to parse the plan. So, why not just return plain text in some service-specific format?

Require the actually executed query is part of the response, though – connecting the plan with the ADQL could be difficult. Example:

```
EXPLAIN SELECT * FROM ppmx.data WHERE cmag<10 LIMIT 2000;

 Limit  (cost=0.00..3105.87 rows=2000 width=151)
   ->  Seq Scan on data  (cost=0.00..629235.88
              rows=405192 width=151)
         Filter: (cmag < 10::double precision)
```

Incidentally, that's an example for why people may want to read a plan: Here, the database has decided not to use an index but rather a sequential scan. Finding out the server has decided so is the first step to fixing queries for which such decisions are bad.

10. Plan XML example

On the other hand, what we're communicating is some annotated expression tree (within the relational algebra). If it's cheap, maybe it's better to mark up that tree even if clients probably won't bother to actually format it?

```
<plan:plan xmlns:plan="http://docs.g-vo.org/std/TAPPlan.xsd"
  <plan:query>
    SELECT * FROM ppmx.data WHERE cmag &lt; 10 LIMIT 2000
  </plan:query>
  <plan:operation>
    <plan:description>Limit</plan:description>
    <plan:rows>
      <plan:value>2000</plan:value>
    </plan:rows>
    <plan:cost>
      <plan:min>50.29</plan:min>
      <plan:max>1975.25</plan:max>
    </plan:cost>
    <plan:operation>
      <plan:description>Bitmap Heap Scan on data</plan:description>
      <plan:rows>
    ...etc
```

So, we have a root element `plan`, containing the query actually executed (i.e., in the native DB dialect), and then operations; eatch operation has a human-readable description, if possible estimates for the cost and/or row count of the query, and possibly subordinate operations.

This stuff works well for postgres. What about other DBMSes?

11. Do you want to know more?

There's a more formal definition of what I'm suggesting at

http://docs.g-vo.org/tapevolution.html

Current plan: Write a note containing these two points together with recommendations on most of the points in TAPImplementationNotes[2]. Who's in?

[2] `http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/TAPImplementationNotes`