Fig. 1

Fig. 2

Fig. 3

1. Feedback on ADQL 2.1

(cf. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(cf. Fig. 2)

Mostly fine: https://blog.g-vo.org/speak-out-on-adql-2-1/

- Optionality and Verification
- The STC-S mess
- BOX.
- Constructors with POINTs
- set operations grammar

- CAST syntax
- TIMESTAMP()?
- boolean_value_expression
- bitwise expressions

(cf. Fig. 3)

2. For the record

I predict we will regret:

- There's now at least 27 features in some way optional. That means we have defined at least 134'217'728 languages. Make things mandatory!
- the grammar is not machine-checked, and there are skeletons in a number of closets.

. . . but yeah, it's too late to do anything fundamental about these.

3. The STC-S Mess

Sect. 3.6.4 has:

> *The text of a REGION value may contain either a simple or a complex spatial region as defined in the STC-S specification.*

Trouble: STC-S is just a note and specifies things like

```
Redshift LSR[K] VELOCITY RADIO 0.3 Error 0.1 0.3 Resolution 0.01
```

We should certainly not import normative content from a note, and STC-S has far too much stuff in it that we can't really deal with in an ADQL context.

Resolution: We need to adopt the Appendix from TAP 1.0. That's what people missing this have implemented against anyway.

4. BOX

BOX is a 2d coordinate interval. People should now even be able to construct it with

```
BOX(a_point, dra, ddel)
```

This is hard to define and hard to implement.

The only advantage over a well-defined ellipsoid is that you'll introduce artefacts the coordinate system. I give you we don't have ellipsoids in ADQL yet. But if we want them, we should define *them* rather than botch around with BOX.

Resolution: Drop BOX(POINT, . . .), and to keep consistency deprecate the whole BOX thing.

5. CIRCLE(POINT, radius)

You can now write `CIRCLE(center, 1)` and `POLYGON(pt1, pt2, pt3)`.

That's not reflected in the grammar yet. Central question: are functions allowed in there?

Resolution: It certainly is cool if people can say `CIRCLE(ivo_apply_pm(ra, dec, pmra, pmdec, +10.4), 1/3600.)` (they can on DaCHS services).

Is that worth the extra effort?

6. Set Operations

When I originally implemented UNION, EXCEPT, and INTERSECT, something in the grammar extension seemed wrong and I took the grammar from postgres.

I forget the issue, don't see it now, and it might have been a simple thinko.

Then again, it might not.

Resolution: Someone use the ADQL 2.1 set operations grammar for an implementation. I won't trust it before that happens. Well, we need a "second" implementation anyway.

7. CAST syntax

CAST(x AS *type*) is defined in the text, but not the grammar.

Question: What can *type* be?

DaCHS currently allows atomic types and stuff like CHAR(20) – but no arrays and no geometries.

Resolution: We have to say what that second argument should minimally be and put that into the grammar.

8. TIMESTAMP?

If we have `CAST(x, TIMESTAMP)`, there's little point in having a separate TIMESTAMP functon.

Resolution:

- Keep it anyway for symmetry with geometries and because CAST is optional?
- Rather simply make cast mandatory?

9. Boolean functions

The current grammar lets you write

```
WHERE bool_fct(a)
```

and

```
WHERE True
```

but not, I think,

```
SELECT * FROM
  (SELECT bool_fct(a) AS x FROM table) as q
WHERE x
```

This is all too confusing.

Resolution: Let's not have more than `x=True` or `f(a)=True` lest things get out of hand.

10. Bitwise operators

The spec wants bitwise operations as operators: `flags&8=8`.
(but it probably gets `flags+4&8|7` wrong)

DaCHS does them as functions: `BITWISE_AND(flags,8)=8`.

Resolution: Let's have functions! Expressions are not *that* more readable. They just need much more grammar we can botch.

11. Break Out?

Can we have a quick breakout meeting with Dave and whoever else can lend a hand to work these out?

Thanks!