Less Cumbersome Units in ADQL

Markus Demleitner
Universität Heidelberg, Astronomisches Rechen-Institut, Mönchhofstraße 12-14, Germany
msdemlei@ari.uni-heidelberg.de

Gefördert durch:
Bundesministerium für Forschung, Technologie und Raumfahrt

Abstract

Since ADQL 2.1, there is a function IN_UNIT that converts a dimensional numeric expression to a user-provided unit; for instance, you could say `IN_UNIT(pmra, 'mas/yr')`. While this already is a valuable tool for writing robust und portable queries, it is a lot to type, and the many letters and special characters make queries hard to read when multiple unit coercions are requested. Also, there is no way to have literals with units in this scheme.

In this contribution, I will propose a more compact syntax that allows one to write literals with units as in `3600{mas/deg}` and to request conversions as in `(rv/2)@{pc/yr}`. I will also discuss the rationale for this design as well as alternatives that might be considered.

Why bother?

Telling the ADQL engine about units will make queries a lot more robust and improve metadata quality. For one, if numbers in expressions carry units, the ADQL engine can annotate the result with the right unit, too.

More importantly, the unit calculus makes expressions more portable and less error-prone when you apply them to other data. For instance, if you write

```
ivo_epoch_prop(ra, dec, IN_UNIT(parallax, 'mas'),
  IN_UNIT(pmra, 'mas/yr'), IN_UNIT(pmdec, 'mas/yr'),
  IN_UNIT(rv, 'km/s'), 2024.4, 2000.0)
```

you will still have to adapt column names and the source epoch when applying your query to a different catalogue, but the machine will take care that once you put these in, the units of the function arguments will be as expected by `ivo_epoch_prop`.

However, with the existing `IN_UNIT`, even relatively simple expressions become hard to read, and there are no provisions for units on literals at all.

The Proposal

To provide more accessible unit facilities in ADQL, we suggest two new ADQL constructs:

- a unit-literal, which is a string of characters not containing curly braces delimited by curly braces (this needs to be a VOUnits string to make sense, but ADQL does not need to know about that)
- the unit-coercion, an operator for which @ seems to be a reasonable choice of terminal.

A unit-literal would be allowed

- after an unsigned-numeric-literal without an operator (semantics: literal with a unit)
- after the unit-coercion operator; the left-hand side of unit-coercion would be a numeric-value-expression (semantics: unit conversion).

I believe this will cover our use cases and makes for a fairly minimal change in the syntax. What this enables is expressions of the type:

```
mass@{solMass} * 1.99e30{kg/solMass}
```

```
SELECT mass@{solMass} * 1.99e30{kg/solMass} AS dynmass,
  ivo_epoch_prop(ra, dec, parallax@{mas},
    pmra@{mas/yr}, pmdec@{mas/yr}, rv@{km/s.},
    2024.4, 2000.0) AS proppos ...
```

Error Conditions

Only unit conversions that retain dimensionality will be supported (if any). That is, this mechanism is no replacement for the proposed `ivo_specconv` UDF.

If a service cannot perform a conversion, either because it does not retain dimensionality (km/s to W/m**2, say), because the service does not have unit information about the expression, or because it cannot parse the input unit into something usable, it is a fatal error; we do not want new sources of silent errors.

A trivial and valid implementation of the proposed mechanism would thus be to simply reject all conversions. But of course, that would not yield a good user experience.

Alternatives

There are several other ways we could do units in ADQL:

- Simply writing units after literals is a non-starter: we would need the VOUnits grammar with all its terminals inside ADQL, and even then we could not distinguish between a unit and a column name from an as-clause without the (optional) AS.
- We could allow unadorned string literals after numeric-value-expression and unsigned-numeric-literal. This probably does not conflict with other legal ADQL constructs, but having distinct characters for unit literals helps producing useful error messages, and we save the need to escape single quotes (which are legal in VOUnits, although for something that is probably useless in our context).
- We could define a fake schema (u, say) that contains the unit literals, so users would write 200*u.km/u.s. That would have the advantage that many users are familiar with it from astropy. But it would in essence require a unit parser of its own, and it is unclear how we would deal with VOUnits functions in this scheme.

Can you try this now?

Frankly: no. I will put it into DaCHS soon after the ADASS, though, if I hear some encouraging words.

If you are confused by all the ADQL on this page, have a look at GAVO's ADQL course, https://docs.g-vo.org/adql. VOUnits 1.1 is doi:10.5479/ADS/bib/2023ivoa.spec.1215G, ADQL 2.1 is doi:10.5479/ADS/bib/2023ivoa.spec.1215M.

Acknowledgement: This work was supported by the e-inf-astro project, BMBF FKZ 05A23VH4