Gaia Data Queries with ADQL

Markus Demleitner *(msdemlei@ari.uni-heidelberg.de)*
Hendrik Heinl *(heinl@ari.uni-heidelberg.de)*

Agenda

- Why bother?
- A first query
- ADQL
- The finer points of TAP

T(able) A(ccess) P(rotocol)

A(stronomical) D(ata) Q(uery) L(anguage)

Open a browser on `http://docs.g-vo.org/adql-gaia/html`

Data Intensive Science

Data-intensive science means:

1. Using many data collections
2. Using large data collections

Point (1) requires standard formats and access protocols to the data, point (2) means moving the data to your box and operating on it with FORTRAN and grep becomes infeasible.

The Virtual Observatory (VO) in general is about solving problem (1), TAP/ADQL in particular about (2).

A First Query

To follow the examples, start TOPCAT and select TAP in the VO menu. Click the pin icon in the upper right corner of the dialog.

In TAP URL: at the bottom of the window, enter `http://gaia.ari.uni-heidelberg.de/tap` and click "Use Service".

At the bottom of the form, at Mode: check “Synchronous” and enter

```
▷ 1    SELECT TOP 1 1+1 AS result FROM gaiadr1.tgas_source
```

in the text box, then click “Ok”. Copying and Pasting from ⟨http://docs.g-vo.org/adql-gaia⟩ is legal.

Why SQL?

The SELECT statement is written in ADQL, a dialect of SQL ("sequel"). Such queries make up quite a bit of the science within the VO.

SQL has been chosen as a base because

- Solid theory behind it (relational algebra)
- Lots of high-quality engines available
- Not Turing-complete, i.e., automated reasoning on "programs" is not very hard

Relational Algebra

At the basis of relational data bases is the relational algebra, an algebra on sets of tuples (“relations”) defining six operators:

- unary *select*
- unary *project*
- unary *rename*
- binary *cartesian product*
- binary *union*
- binary *set difference*

Good News: You don’t *need* to know any of this.

SELECT for real

ADQL defines just one statement, the `SELECT` statement, which lets you write down expressions of relational algebra. Roughly, it looks like this:

`SELECT` [`TOP` *setLimit*] *selectList* `FROM` *fromClause* [`WHERE` *conditions*] [`GROUP BY` *columns*] [`ORDER BY` *columns*]

TOP

setLimit: just an integer giving how many rows you want returned.

▷ 2 `SELECT TOP 5 * FROM gaiadr1.tgas_source`

▷ 3 `SELECT TOP 10 * FROM gaiadr1.tgas_source`

SELECT: ORDER BY

ORDER BY takes *columns*: a list of column names (or expressions), and you can add ASC (the default) or DESC (descending order):

▷ 4
```
SELECT TOP 5 source_id, parallax
FROM gaiadr1.tgas_source
ORDER BY parallax
```
▷ 5
```
SELECT TOP 5 source_id, parallax
FROM gaiadr1.tgas_source
ORDER BY parallax DESC
```
▷ 6
```
SELECT TOP 5 source_id, phot_g_mean_mag , parallax
FROM gaiadr1.tgas_source
ORDER BY phot_g_mean_mag, parallax
```
Note that ordering is outside of the relational model.

SELECT: what?

The select list has column names or expressions involving columns.

SQL expressions are not very different from those of other programming languages.

▷ 7

```
SELECT TOP 10
  source_id,
  SQRT(POWER(pmdec_error,2)+POWER(pmra_error,2)) AS
pm_errTot
FROM gaiadr1.tgas_source
```

Use `COUNT(*)` to figure out how many items there are.

▷ 8

```
SELECT count(*) AS numEntries
FROM gaiadr1.tgas_source
```

SELECT: WHERE clause

Behind the WHERE is a logical expression; these are similar to other languages as well, with operators `AND`, `OR`, and `NOT`.

▷ 9

```
SELECT source_id, ra, dec
  FROM gaiadr1.tgas_source
  WHERE
  phot_g_mean_flux > 13
  AND parallax < 0.2
```

SELECT: Grouping

For histogram-like functionality, you can compute factor sets, i.e., subsets that have identical values for one or more columns, and you can compute aggregate functions for them.

▷ 10
```
SELECT COUNT(*) AS n,
  ROUND(phot_g_mean_mag) AS bin,
  AVG(parallax) AS parallax_mean
FROM gaiadr1.tgas_source
GROUP BY bin
ORDER BY bin
```

For simple GROUP applications, you can shortcut using DISTINCT (which basically computes the “domain”).

▷ 11
```
SELECT DISTINCT
ROUND(phot_g_mean_mag), ROUND(parallax)
FROM gaiadr1.tgas_source
```

SELECT: JOIN USING

The tricky point in ADQL is the `FROM` clause. So far, we had a single table. Things get interesting when you add more tables: JOIN.

▷ 12

```
SELECT TOP 10 h1.ra, h1.dec, h1.hip, t1.hip
FROM hipparcos AS h1
JOIN tycho2 AS t1
USING (hip)
```

`JOIN` is a combination of cartesian product and a select.

```
FROM hipparcos AS h1
JOIN tycho2 AS t1
USING (hip)
```

yields the cartesian product of the hipparcos and tycho2 tables but only retains the rows in which the hip columns in both tables agree.

SELECT: JOIN ON

If your join criteria are more complex, you can join `ON`:

▷ 13
```
SELECT TOP 20 source_id, h.hip
FROM gaiadr1.tgas_source AS tgas
LEFT OUTER JOIN hipparcos as h ON (tgas.phot_g_mean_mag
BETWEEN
h.hpmag -0.05 AND h.hpmag+0.05)
```

- `t1 INNER JOIN t2`
- `t1 LEFT OUTER JOIN t2`
- `t1 RIGHT OUTER JOIN t2`
- `t1 FULL OUTER JOIN t2`

Geometries

The main extension of ADQL wrt SQL is addition of geometric functions.

Keep the crossmatch pattern somewhere handy (everything is in degrees):

▷ 14

```
SELECT TOP 5
  source_id, tgas.ra, tgas.dec, tm.raj2000,
        tm.dej2000, hmag, e_hmag
FROM gaiadr1.tgas_source as tgas
JOIN twomass AS tm
ON 1=CONTAINS (
     POINT('ICRS', tm.raj2000, tm.dej2000),
     CIRCLE('ICRS', tgas.ra, tgas.dec, 1.5/3600))
```

Subqueries

One of the more powerful features of SQL is that you can have subqueries instead of tables within FROM. Just put them in parentheses and give them a name using AS. This is particularly convenient when you first want to try some query on a subset of a big table:

▷ 15
```
SELECT count(*) as n, round((hmag-jmag)*2) as bin
FROM (
  SELECT TOP 4000 * FROM twomass) AS q
GROUP BY bin
ORDER BY bin
```

Subqueries on Gaia Xmatchtables

Change the TAP service to

`http://gea.esac.esa.int/tap-server/tap`

In the query form type:

▷ 16
```
SELECT *
FROM tmass_original_valid AS tmov
JOIN
  (SELECT tgas.*, tmbn.tmass_oid
   FROM gaiadr1.tgas_source AS tgas
   JOIN tmass_best_neighbour AS tmbn ON
        tgas.source_id=tmbn.source_id
   WHERE 1=CONTAINS(POINT('ICRS', tgas.ra, tgas.dec),
        CIRCLE('ICRS', 189.2, 62.21, 1.0))) AS tgtm
USING (tmass_oid)
```

TAP: Uploads

TAP lets you upload your own tables into the server for the duration of the query.

Example: Take a subset of tgas_source with positions and proper motions and crossmatch it with sdss to get colors. First we make the subset with:

▷ 17

```
SELECT TOP 200
source_id, ra, dec, pmra, pmdec
FROM gaiadr1.tgas_source
WHERE 1=CONTAINS(POINT('ICRS', raj2000, dej2000),
      CIRCLE('ICRS', 18.02, 9.281, 4.0 ))
```

TAP: Uploads 2

Then we change the TAP Service to `http://dc.zah.uni-heidelberg.de/tap` and perform the following query:

▷ 18

```
SELECT TOP 100
    tgas.*, sdss.u, sdss.i, sdss.r, sdss.g
    FROM sdssdr7.sources AS sdss
    JOIN TAP_UPLOAD.t1 AS tgas
    ON 1=CONTAINS(
          POINT('ICRS', sdss.ra, sdss.dec),
          CIRCLE('ICRS', tgas.ra, tgas.dec, 3./3600.))
```