A Short Course On ADQL

Markus Demleitner Hendrik Heinl

July 25, 2024

GAVO

TAP/ADQL

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

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

Open a browser on http://docs.g-vo.org/adql for lecture notes.

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.

At *Keywords*, type **gavo**. Wait until the results are filtered and select the entry *GAVO DC TAP*. Then click *Use Service*.

In the query pane, enter:

```
SELECT TOP 1 1+1 AS result FROM ivoa.obscore
```

and then click “Ok”.

You can also use TAP from Python. A lot more on this later. If you are curious now, see an Ψipython notebook explaining the basics.

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") plus 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 only 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: an integer giving how many rows you want returned.

```
SELECT TOP 5 * FROM rave.main
```

```
SELECT TOP 10 * FROM rave.main
```

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):

```
SELECT TOP 5 *
FROM rave.main
ORDER BY rv

SELECT TOP 5 *
FROM rave.main
ORDER BY rv DESC

SELECT TOP 5 *
FROM rave.main
ORDER BY fiber_number, rv
```

Note that `SELECT *` (pulling all columns) is usually wasteful and you should do better from the next slide on.

Also 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.

```
SELECT TOP 10
  POWER(10, phot_g_mean_mag) AS rel_flux,
  SQRT(POWER(ra_error, 2)+POWER(dec_error, 2)) AS errTot
FROM gaia.dr3lite
```

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

```
SELECT count(*) AS numEntries FROM rave.main
```

SELECT: WHERE clause

Behind the WHERE is a logical expression; these are similar to other languages as well, with boolean operators `AND`, `OR`, and `NOT`. To find bright stars (apparently) moving quickly towards or from us:

```
SELECT raveid FROM rave.main
WHERE
  jmag<10
  AND ABS(rv)>100
```

Missing Data: NULLs

SQL has an explicit concept of missing data: The magic value NULL. It has some interesting properties:

```
SELECT count(*) FROM tap_schema.tables WHERE NULL=NULL
```

returns 0. So does

```
SELECT count(*) FROM tap_schema.tables WHERE NULL!=NULL
```

All comparisons with NULLs are false.

To select rows for which a given piece of data is or is not NULL use the special construct `IS (NOT) NULL`.

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.

```
SELECT
  COUNT(*) AS n,
  ROUND(mv) AS bin,
  AVG(color) AS colav
FROM dmubin.main
GROUP BY bin
ORDER BY bin
```

To just figure out the domain of columns, there is a shortcut: `DISTINCT`.

SELECT: Grouping by HEALPix

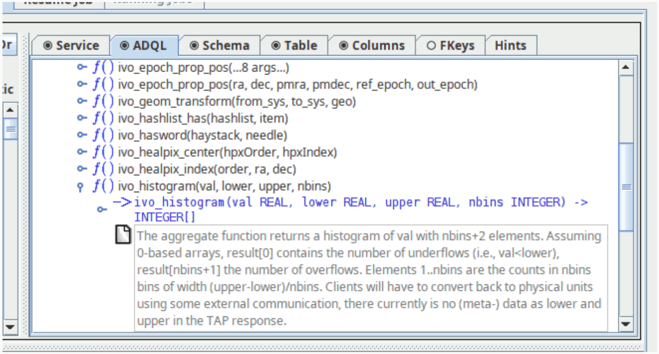
If you want to characterise some property over the sky, HEALPixes are your friend.

```
SELECT ivo_healpix_index(5, raj2000, dej2000) AS bin,
  COUNT(*) AS n,
  AVG(rv) AS meanrv,
  MAX(rv)-avg(rv) AS updev,
  AVG(rv)-min(rv) AS lowdev
FROM rave.main
WHERE e_rv<20
GROUP BY bin
HAVING COUNT(*)>5
```

ADQL User Defined Functions

`ivo_healpix_index` is an example of an ADQL extension mechanism: Operators can add *UDF*s.

See TOPCAT's ADQL TAP for the UDFs available on a service:



```
f() ivo_epoch_prop_pos(...8 args...)
f() ivo_epoch_prop_pos(ra, dec, pmra, pmdec, ref_epoch, out_epoch)
f() ivo_geom_transform(from_sys, to_sys, geo)
f() ivo_hashlist_has(hashlist, item)
f() ivo_hasword(haystack, needle)
f() ivo_healpix_center(hpxOrder, hpxIndex)
f() ivo_healpix_index(order, ra, dec)
f() ivo_histogram(val, lower, upper, nbins)
    ->ivo_histogram(val REAL, lower REAL, upper REAL, nbins INTEGER) ->
    INTEGER[]
```

The aggregate function returns a histogram of val with nbins+2 elements. Assuming 0-based arrays, result[0] contains the number of underflows (i.e., val<lower), result[nbins+1] the number of overflows. Elements 1..nbins are the counts in nbins bins of width (upper-lower)/nbins. Clients will have to convert back to physical units using some external communication, there currently is no (meta-) data as lower and upper in the TAP response.

SELECT: JOIN USING

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

```
SELECT TOP 10 lat, long, flux
FROM lightmeter.measurements
JOIN lightmeter.stations
USING (stationid)
```

JOINing is Selecting from the Cartesian Product

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

`measurements JOIN stations USING (stationid)`
yields the cartesian product of the measurement and stations tables but only retains the rows in which the stationid columns in both tables agree.

$A = \{(a, 1), (b, 2), (b, 3)\}$
$B = \{(1, u), (2, v)\}$
$A \times B =$

(a,	1,	1,	u)
(a,	1,	2,	v)
(b,	2,	1,	u)
(b,	2,	2,	v)
(b,	3,	1,	u)
(b,	3,	2,	v)

SELECT: JOIN ON

If your join criteria are more complex than simple equality, you can join ON.

```
SELECT dateobs as lswdate, t_min as appdate
FROM lsw.plates AS a
LEFT OUTER JOIN applause.main AS b
ON (dateobs BETWEEN t_min AND t_max)
WHERE dateobs BETWEEN 36050 and 36100
```

Flavours of JOIN

There are various kinds of joins, depending on what elements of the cartesian product are being retained in the presence of missing data (NULL).

- 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.

```
SELECT TOP 500 rv, e_rv, p.radial_velocity,
 p.ra, p.dec, p.pmra, p.pmdec
FROM gaia.dr3lite AS p
JOIN rave.main AS rave
ON 1=CONTAINS(
 POINT(p.ra, p.dec),
 CIRCLE(rave.raj2000, rave.dej2000, 1.5/3600.))
```

There are more geometry functions defined in ADQL:

`AREA, BOX, CENTROID, CIRCLE, CONTAINS, COORD1, COORD2, COORDSYS, DISTANCE, INTERSECTS, POINT, POLYGON`

DISTANCE

ADQL has a `DISTANCE` function to compute the spherical distance between two points:

```
DISTANCE(lon1, lat1, lon2, lat2)
```

The `DISTANCE` function can be used to make cone selections and is the prefered way to perform crossmatches on sky positions in ADQL 2.1.

```
SELECT TOP 1000
  raj2000, dej2000, parallax
  FROM arihip.main
  WHERE
    DISTANCE(raj2000, dej2000,
             189.2, 62.21) < 10
```

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:

```
SELECT COUNT(*) AS n, ROUND((u-z)*2) AS bin
FROM (
  SELECT TOP 4000 * FROM sdssdr16.main) AS q
GROUP BY bin ORDER BY bin
```

Common table expressions

`WITH` lets you name a subquery result for later use in your main query.

```
WITH withrvs AS (SELECT TOP 200
  ra, dec, source_id,
  a.radial_velocity, b.rv as raverv
  FROM gaia.dr3lite AS a
  JOIN rave.main AS b
  ON (
    DISTANCE(a.ra, a.dec,
      b.raj2000, b.dej2000) < 1/3600.))
SELECT *
FROM gdr3spec.spectra
JOIN withrvs
USING (source_id)
```

TAP: Uploads

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

Example: Add proper motions to an object catalogue giving positions reasonably close to ICRS; grab some table, falling back to the attached Ψex.vot, load it into TOPCAT, go to the TAP window and there say:

```
SELECT mine.*, refcat.pmra, refcat.pmde FROM
  gaia.dr3lite AS refcat
  JOIN tap_upload.t1 AS mine
  ON DISTANCE (
    refcat.ra, refcat.dec,
    mine.raj2000, mine.dej2000) < 0.001
```

Almost real world

Suppose you have a catalogue giving alpha, delta, and an epoch of observation sufficiently far away from the Gaia epoch. To match it, you have to bring the reference catalogue on our side to the epoch of your observation.

```
SELECT alpha, delta, parallax, pmra, pmdec, source_id
FROM (
SELECT
  alpha, delta, parallax, pmra, pmdec, source_id,
  ivo_epoch_prop_pos(ra, dec, parallax,
    pmra, pmdec, radial_velocity, 2016, epoch) AS tpos
FROM tap_upload.t1
  JOIN gaia.dr3lite
  ON DISTANCE(alpha, delta, ra, dec)<0.1) AS q
WHERE DISTANCE(POINT(alpha, delta), tpos)<2/3600.
```

TAP: Async operation

TAP jobs can take hours or days. To support that, you can run your TAP jobs asynchronously. This means you do not have to keep a connection open all the time.

To go async in TOPCAT, change the *Mode* selector to "*Asynchronous*". After submitting the job, you can watch your job go through "*UWS* phases":

PENDING	Job created, you can configure it
QUEUED	Waiting for compute time
EXECUTING	The job is running
COMPLETED	Successful completion, fetch results
ERROR	The Job has failed, fetch error message

Resuming async Jobs

You can quit your client with async and resume from somewhere else.

To do that: In *Running Jobs*, select the URL and save it. Uncheck *Delete on Exit* and leave TOPCAT.

Then restart TOPCAT, open the TAP window and paste the URL back into the URL field. If the job has finished, you can retrieve the result.

TAP: the TAP schema

TAP services try to be self-describing about what data they contain. They provide information on what tables they contain in special tables in *TAP_SCHEMA*. Figure out what tables are in there by querying *TAP_SCHEMA* itself:

```
SELECT * FROM tap_schema.tables
WHERE table_name LIKE 'tap_schema.%'
```

To see what columns there are in *tap_schema.columns*, say:

```
SELECT * FROM tap_schema.columns
WHERE table_name='tap_schema.columns'
```

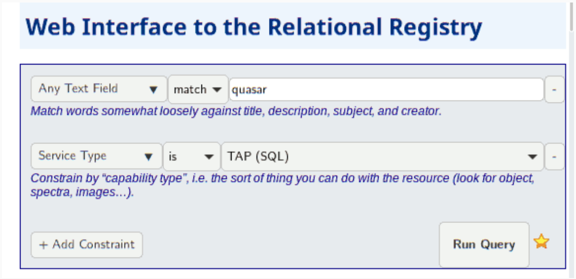
Of course, in normal operations, clients like TOPCAT do that querying for you: it's how they fill their metadata views.

Data Discovery 1: the Registry

The list of services in TOPCAT's TAP window comes from the VO Registry, an inventory of the services and data kept within the VO.

There are a few more ways to search the Registry, for instance in a web browser using WIRR.

Use case: Find tables talking about quasars that have redshifts.



Web Interface to the Relational Registry

Any Text Field ▾ match ▾ quasar

Match words somewhat loosely against title, description, subject, and creator.

Service Type ▾ is ▾ TAP (SQL) ▾

Constrain by "capability type", i.e. the sort of thing you can do with the resource (look for object, spectra, images…).

Data Discovery 2: use ADQL

The relational registry standard says how to query this data set using ADQL. All tables are in the rr schema and can be combined through NATURAL JOIN. Our use case looks like this in ADQL:

```
SELECT ivoid, access_url, name,
  ucd, column_description
FROM rr.capability
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.table_column
  NATURAL JOIN rr.res_table
WHERE standard_id LIKE 'ivo://ivoa.net/std/tap%'
  AND 1=ivo_hasword(table_description, 'quasar')
  AND ucd='src.redshift'
```

Simbad

Simbad has a TAP interface; find it TOPCAT's server selector and inspect Simbad's table metadata. Try queries like:

```
SELECT TOP 20 * FROM basic
```

```
SELECT TOP 1000
   otype_txt, tc.*
   FROM basic AS db
   JOIN TAP_UPLOAD.t7 AS tc
   ON 1=CONTAINS(POINT('ICRS', db.ra, db.dec),
                CIRCLE('ICRS', tc.ra, tc.dec, 2./3600.))
   WHERE otype_txt!='star'
```

Onward

If you get stuck or a query runs forever, the operators are usually happy to help you. To find out who could be there to help you, check TOPCAT's Service tab or use – the relational registry. If you have the ivoid of the service, say

```
SELECT role_name, email, base_role
FROM rr.res_role
WHERE ivoid='ivo://org.gavo.dc/tap'
```

– if all you have is the access URL, do a natural join with interfaces.

If we have done a good job, you now know how. . .