Fig. 1

1. ADQL

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

Agenda

- ORDER (why bother?)
- WHERE (where to enter ADQL)
- SELECT (the SELECT statement)
- REGION (ADQL's geometric functions)
- FROM (where can you go from here)

First steps in the

Astronomical

Data

Query

Language

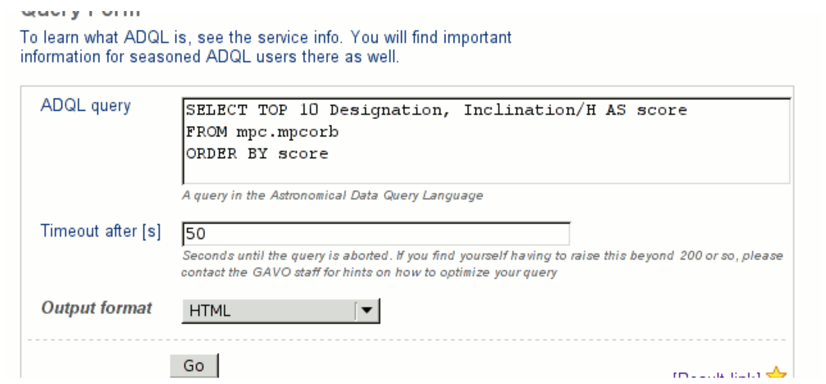
2. Why bother?

Moving multi-terabyte or petabyte data sets is not an option in astronomy. Even less when you want two of them.

(vgl. Fig. 1)

Thus: You will have to move your smarts to where some data happens to be. In addition to smart planning (so only the data actually required gets transferred), you need a language to express your smarts. It's not going to be FORTRAN. Maybe it's going to be ADQL.

In case you're outraged: Well, in theory it could even become FORTRAN – who can tell with all the cloud hype that's going on. The trouble with FORTRAN is that it's hard to reason about what a FORTRAN program is going to do, which is why it takes a bit of courage to let people upload and execute FORTRAN programs on your box.



To learn what ADQL is, see the service info. You will find important information for seasoned ADQL users there as well.

ADQL query

```
SELECT TOP 10 Designation, Inclination/H AS score
FROM mpc.mpcorb
ORDER BY score
```

A query in the Astronomical Data Query Language

Timeout after [s]: 50

Seconds until the query is aborted. If you find yourself having to raise this beyond 200 or so, please contact the GAVO staff for hints on how to optimize your query

Output format: HTML

Fig. 2

3. Let me play

To try out ADQL, go to GAVO's web ADQL form[1] and type in some SQL (if you know any). The tables you can query are under the link on the right („Tables available for ADQL").

Here's the query shown in the talk:

▷ (1)
```
SELECT TOP 10 Designation, Inclination/H AS score
FROM mpc.mpcorb
ORDER BY score
```

Can't you almost read it?

If you enter this and nobody has posted a query against this table shortly before, you'll probably get an error message to the effect that a timeout occurred. Increase the timeout length – the query shouldn't take all that long.

There's an HTML link sheet floating around somewhere that saves you the trouble of typing in or cutting and pasting what's in here.

(vgl. Fig. 2)

4. 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* – select tuples matching to some condition
- unary *project* – make a set of sub-tuples of all tuples (i.e., have less columns)
- unary *rename* – change the name of a relation (this is a rather technical operation)
- binary *cartesian product* – the usual cartesian product, except that the tuples are concatenated rather than just put into a pair; this, of course, is not usually actually computed but rather used as a formal step.
- binary *union* – simple union of sets. This is only defined for "compatible" relations; the technical points don't matter here
- binary *set difference* as for union; you could have used intersection and complementing as well, but complementing is harder to specify in the context of relational algebra

Good News: You don't *need* to know any of this. It's just reassuring to know that there's a solid theory behind all of this. And I wanted to give some good news.

[1] http://vo.uni-hd.de/adql

5. 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*]

In reality, there are yet a few more things you can write, but what's shown covers most things you'll want to do. The real magic is in *selectList*, *fromClause* (in particular), and *conditions*.

TOP

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

▷ (2) `SELECT TOP 5 * FROM rave.main`

▷ (3) `SELECT TOP 10 * FROM rave.main`

The typewriter text with the triangles in front of it are the examples shown in the talk. Try entering them on our ADQL page[2] and play with them.

ORDER BY

columns: a list of column names (or expressions).

▷ (4) `SELECT TOP 5 name, rv FROM rave.main ORDER BY rv`

▷ (5) `SELECT TOP 5 name, rv FROM rave.main ORDER BY rv DESC`

▷ (6) `SELECT TOP 5 name, fiber, rv FROM rave.main ORDER BY fiber, rv`

Note that ordering is outside of the relational model; that sometimes matters because it may mess up "query planning", a rearrangement of relational expressions done by the database engine to make them run faster.

6. SELECT: what?

select list

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
  POWER(10, alfa_Fe) AS ppress,
  SQRT(SQUARE(e_pmde)+SQUARE(e_pmra)) AS errTot
FROM rave.main
```

The usual operators work, you have single-quoted strings, functions you can use include:

ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, DEGREES, EXP, FLOOR, LOG, LOG10, MOD, PI, POWER, RADIANS, RAND, ROUND, SIN, SQUARE, SQRT, TAN, TRUNCATE.

Also note how I used AS to rename a column. You can use the names assigned in this way in, e.g., ORDER BY:

▷ (8)
```
SELECT TOP 10
  POWER(10, alfa_Fe) AS ppress,
  SQRT(SQUARE(e_pmde)+SQUARE(e_pmra)) AS errTot
FROM rave.main
ORDER BY ppress
```

To select all columns, use *

▷ (9) `SELECT TOP 10 * FROM rave.main`

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

▷ (10) `SELECT count(*) AS numEntries FROM rave.main`

[2] http://vo.uni-hd.de/adql

WHERE clause

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

▷ (11)
```
SELECT name FROM rave.main
WHERE
  obsDate>'2005-02-02'
  AND imag<12
  AND ABS(rv)>100
```

7. SELECT: Whence

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

Check the table structures under Tables available: flux is from measurements, lat and long from stations.

JOIN is a combination of cartesian product and a select. When you write
`measurements JOIN stations USING (stationid)`

you get the cartesian product of the measurement and stations tables but only retain the rows in which stationid is the same.

▷ (13)
```
SELECT TOP 10 *
FROM lightmeter.measurements
JOIN lightmeter.stations
USING (stationid)
```

Note that while the stationid column we're joining on is in both tables but only occurs once in the joined table.

If your join criteria are more complex, you can join ON:
```
dmubin LEFT OUTER JOIN rave ON
  (dmubin.mv BETWEEN rave.imag-1 AND rave.imag+1)
```

▷ (14)
```
SELECT TOP 10 *
FROM dmubin.main AS dmu
LEFT OUTER JOIN rave.main AS rave
ON (dmu.mv BETWEEN rave.imag-1 AND rave.imag+1)
```

With outer joins, the criteria in the select from the cartesian product can be influenced. Play with the example and read some SQL textbook (e.g., the PostgreSQL docs)

8. 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.

▷ (15)
```
SELECT
  COUNT(*) as n,
  ROUND(mv) AS bin,
  AVG(color) AS colav
FROM dmubin.main
GROUP BY bin
ORDER BY bin
```

ADQL's aggregate (set) functions include `AVG`, `MAX`, `MIN`, `SUM`, `COUNT`.

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

▷ (16)
```
SELECT DISTINCT comp, FK FROM dmubin.main
```

9. Geometries

The main extension of ADQL wrt SQL is addition of "geometric" functions. Unfortunately, these were not particularly well designed, but if you don't expect too much, they'll do their job.

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

```
1=CONTAINS(
  POINT('ICRS', bigcatRA, bigcatDE),
  CIRCLE('ICRS', smallcatRA, smallcatDE, matchradius))
```

▷ (17)
```
SELECT rv, e_rv,
 p.raj2000, p.dej2000, p.pmRA, p.pmDE
FROM ppmxl.main AS p
JOIN rave.main AS rave
ON 1=CONTAINS(
  POINT('ICRS', rave.raj2000, rave.dej2000),
  CIRCLE('ICRS', p.raj2000, p.dej2000, 1.5/3600.))
```

In theory, you could use various systems and hope the server converts the positions, but I'd avoid constructions with multiple systems – even if the server implements the stuff correctly, it's most likely going to be slow.

Some sites have extra features in the `REGION` construct. On my site, you can, e.g., say `REGION('simbad <obj>')`.

▷ (18)
```
SELECT raj2000, dej2000, pmRA, pmDE
FROM ppmxl.main
WHERE 1=CONTAINS(
  REGION('simbad M1'),
  CIRCLE('ICRS', raj2000, dej2000, 0.05))
```

10. Where do we go from here?

- Get the lecture notes for this talk from http://www.g-vo.org and play around.
- Read up on SQL – there a some ok books out there and many bad ones. The PostgreSQL docs[3] are nice, too.
- Read the service info of the DC's ADQL service; there's going to be news there now and then.
- **The Future:** TAP.

TAP, the Table Access Protocol, lets you submit long-running queries, inspect table structures in a standardized way, and upload your own tables. The standard is new, and current implementations are shaky and incomplete, but that is going to change. When our TAP service is ready for some beating, it's going to be news on the DC ADQL page.

[3] `http://www.postgresql.org/docs/`