Using TAP to query Gaia gog data

Markus Demleitner, Hendrik Heinl

September 16, 2015

Abstract

In this brief tutorial you will learn to use the table access protocol (TAP) and the astronomical fata query language (ADQL) to access the Gaia gog data. The actual gaia data will be accessed analogously. As client we will be using TOPCAT. At the end of the tutorial you should not only know how to access the gaia gog data, but also have some insides in the usefulness of standards and the Virtual Observertory in particular. As example for using remote services we will remotely calculate the mean proper motions of stars in selected nearby galaxies

CAVE: Though we expect to find stars belonging to nearby galaxies in the Gaia gog data, this data just is simulated data, so you are not supposed to do real science on it. But of course you are welcome to use it to prepare your scripts and data for the real Gaia data releases.

Software: TOPCAT Version 4.3, TAP

1 Querying SIMBAD

▷ **1** *Querying SIMBAD* – In a first step we will be querying SIMBAD to get the positions and angular size nearby galaxies. Therefore we start TOPCAT and open the TAP Query window at VO →TAP. In the field *Keywords* enter "SIMBAD" and click Find Service. From the two results showing up click on "SIMBAD TAP". Then we click Use Service. The new opening window shows on the left a list of all available tables. Thanks to the *"Find:"* feature we don't need to scroll down but can directly type "basic" into this field and then check the table "public.basic" on which we will run our query. In the query field we write our ADQL query as following:

```
SELECT TOP 20
ra, dec, galdim_majaxis, main_id
FROM public.basic
WHERE otype='G..'
AND galdim_majaxis IS NOT NULL
AND galdim_majaxis <100
ORDER BY galdim_majaxis DESC
```

As you see this is quite straightforward: in the first line we ask for the first 20 results (due to latency time in tutorials we limit this to 20). Line two shows the columns that we want in return, which are coordinates, the angular size of the object and the main indentifier. "FROM" defines the table we want to query and after "WHERE" we define the conditions. From the metadata we see that "otypes" is the column in which the calue "G.." is used for identified galaxies. As an additional condition we don't want those galaxies returned that don't have an angular size, or those which are too large - the latter is due to the expected performance loss when querying the Gaia data in a tutorial. The last line means we want the data to be sorted by descending angular size. Click Run Query and receive the results as a table.

▷ **2** *Local data* – In TOPCATs main window you now find the table with the query results. Click on Views →Table Data to take a look at the data. We now could reduce this data by deleting all the galaxies that we don't need, and for your research you may find that very usefull, but for the next step we don't need to do this.

2 Gaia TAP service

Now we are ready to use our local data to query the Gaia TAP service. We want to let the TAP service do as much of the process as possible to just download only the data we are interested in. Therefore we again open the TOPCAT TAP query Window as we did before, but instead of SIMBAD we now use the TAP URL: http://mintaka.ari.uni-heidelberg.de/tap/

▷ **3** *Building the query* – In the *Use Service* Tab we chose table *gog.gdr001* which contains the simulated data. By browsing of the Columns Metadata we see that to calculate the mean proper motion we need the proper motion components from declination and right ascension directions, which are the columns *pmra, pmdec*. From our own data we want *main_id* and finally we want to count the

stars belonging to the galaxy as defined in column main_id. We start by pressing on Examples → Basic → Fulltable

```
SELECT TOP 1000 * FROM gog.gdr001
```

This is a good start for the query. But actually we are not interested in all the data, so we change the query to our needs: We want to calculate the proper motion from the existing columns pmra and pmdec. This calculation we can do remotely on the TAP service so we should do so by adding the calculation into our TAP query as following:

```
SELECT TOP 25000
AVG(sqrt(pmra*pmra+pmdec*pmdec)) AS pmtot,
COUNT(*) AS ct, main_id
```

▷ **4** *Upload local data* – Now we need to add our local data as well as define the remote table we want to query. We use the feature of a *tap_upload* to sent our local data to the TAP service. Here we have to be aware that we set the right identifier of our local table. In TOPCAT that is the table number, so .t1 would take the first table in TOPCAT. We also want to benefit from aliases, so we define the alias of our data as "mine" and the remote data as "theirs". As we want to merge some of the local data with the remote data we use *JOIN ... ON*. So the next lines are:

```
FROM tap_upload.t1 AS mine
JOIN gog.gdr001 AS theirs ON
```

▷ **5** *Query Conditions* – Now we have to set the conditions of our query. What we want are stars from the remote service in a cone defined by the coordinates and the size of the galaxies in arcminutes as we retrieved it from SIMBAD. In ADQL we do this like this:

```
(1=CONTAINS(POINT('icrs', theirs.ra, theirs.dec),
CIRCLE('icrs', mine.ra, mine.dec, mine.galdim_majaxis/60)))
```

Take a minute to understand what's happening here. As a hint it's helpful to know that CONTAINS() returns a boolean as "1" - true or "0" - false. Note how usefull setting the namespace was for this. In a final step we add a grouping so the counted stars are mapped to *main_id* and we order the data in descending order of column ct. The final query is:

```
SELECT
```

```
AVG(SQRT(pmra*pmra+pmdec*pmdec)) AS pmtot,
COUNT(*) AS ct, main_id
FROM tap_upload.t1 AS mine
JOIN gog.gdr001 AS theirs
ON (1=CONTAINS(POINT('icrs', theirs.ra, theirs.dec),
CIRCLE('icrs', mine.ra, mine.dec, mine.galdim_majaxis/60)))
GROUP BY main_id
ORDER BY ct DESC
```

Write this into the Query field and click *Run Query*. Due to the calculation the result may take some time to return.