

An Introduction to ADQL

Yaguang Li (University of Hawaii)

Why an ADQL tutorial?

- How to cross match my own catalog with Gaia sources?
- How to select some stars from Gaia sources?
- A missing course in astronomy for accessing large databases.
- Could be (self-)learned in 1 hour!



Advanced positional cross-match

```
SELECT dr3.source_id, dr3.ra, dr3.dec, dr3.pmra,
dr3.pmra_error, dr3.pmdec, dr3.pmdec_error, dr3.ruwe, gspc.c_star,
gspc.u_jkc_mag, gspc.u_jkc_flux, gspc.u_jkc_flux_error, gspc.u_jkc_flag,
gspc.b_jkc_mag, gspc.b_jkc_flux, gspc.b_jkc_flux_error, gspc.b_jkc_flag,
gspc.v_jkc_mag, gspc.v_jkc_flux, gspc.v_jkc_flux_error, gspc.v_jkc_flag,
gspc.y_ps1_mag, gspc.y_ps1_flux, gspc.y_ps1_flux_error, gspc.y_ps1_flag,
gspc.f606w_acswfc_mag, gspc.f606w_acswfc_flux,
gspc.f606w_acswfc_flux_error, gspc.f606w_acswfc_flag
FROM gaiadr3.gaia_source AS dr3
JOIN gaiadr3.synthetic_photometry_gspc AS gspc USING (source_id)
WHERE
CONTAINS(
POINT(COORD1(EPOCH_PROP_POS(201.697,-47.479472,
0.1368,-3.2400,-6.7300,234.2800,2000,2016.0))),
COORD2(EPOCH_PROP_POS(201.697,-47.479472,
0.1368,-3.2400,-6.7300,234.2800,2000,2016.0))),
CIRCLE(ra, dec, 1.))=1
```

Positional cross-match + proper-motion propagation

```
SELECT subquery.source_id, subquery.parallax,
subquery.phot_g_mean_mag, galex.*
FROM (
SELECT gaia.*, EPOCH_PROP_POS(ra, dec, parallax, pmra, pmdec,
radial_velocity, ref_epoch, 2000)
AS propagated_position_vector
FROM gaiadr3.gaia_source AS gaia
WHERE 1 = CONTAINS(
POINT(56.75, 24.12),
```

Platform

https://gea.esac.esa.int/archive/

Signing-in allows you to upload your own tables.
Not necessary for today's tutorial.

The screenshot shows the Gaia Archive website interface. At the top, there is a header with the European Space Agency (ESA) logo and the text "gaia archive". Below the header is a navigation bar with links: HOME, SEARCH, SINGLE OBJECT, VISUALISATION, and HELP. The "SEARCH" link is highlighted. Below the navigation bar, there are tabs for "Basic", "Advanced (ADQL)", and "Query Results". The "Advanced (ADQL)" tab is selected. On the left side, there is a sidebar with a search bar containing the text "gaia" and a list of categories: Other, Gaia Data Release 1, Gaia Data Release 2, Gaia Data Release 3, Astrophysical parameters, Auxiliary, Cross match, Extra-galactic, Non-single stars, Performance verification, Reference frame, Science alerts, Simulation, Solar system, Spectroscopy, Variability, and Gaia Early Data Release 3. The main content area has a "Job name:" field, a large text input area for the ADQL query, and buttons for "Reset Form" and "Submit Query". Below the input area, it says "Ctrl+Space for query autocompletion". At the bottom, there is a table with the header "No results found" and columns: Status, Job, Creation date, Num. rows, and Size. Annotations include an orange arrow pointing to the "SIGN IN" link in the top right corner, two orange arrows pointing to the "Basic" and "Advanced (ADQL)" tabs, and an orange arrow pointing to the large text input area for the ADQL query.

Put ADQL code here.

Platform

Topcat: VO -> Table Access Protocol (TAP) Query

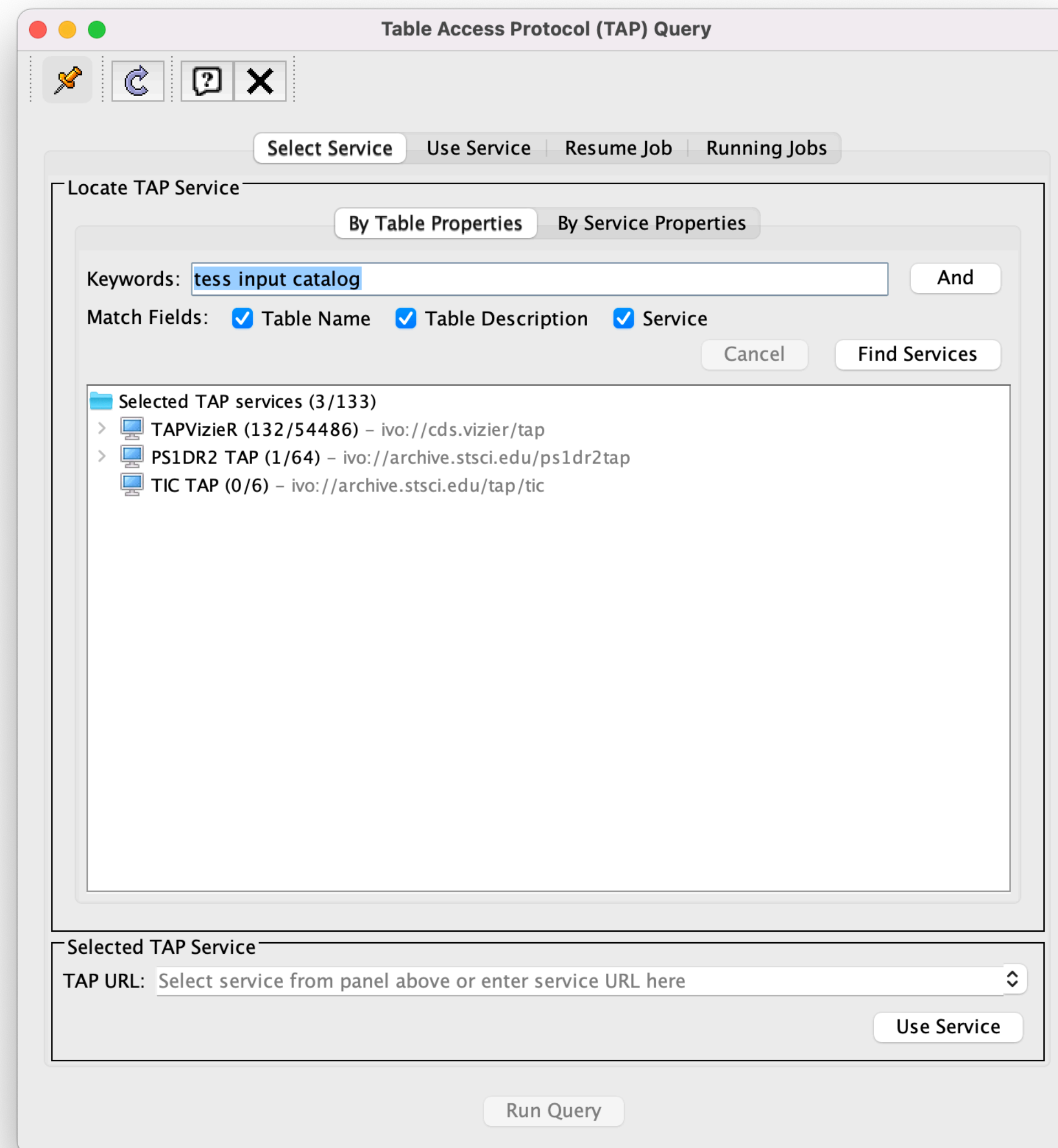


Table of contents

1. Introduction
2. Basic query structure
3. Common ADQL functions and operators
4. Spatial queries in ADQL
5. Joins and subqueries
6. Best practices and tips
7. Resources and further learning

1. Introduction

What is ADQL?

- Virtual observatory (VO) is a database that stores a collection of relations (tables).
- To extract information from a database, we construct a *query*.
 - Astronomical Data Query Language (ADQL)
 - a variant of the Structured Query Language (SQL)
- Access is done via the Table Access Protocol (TAP).

2. Basic query structure

SELECT/FROM/TOP statement

- Syntax ([] means optional)

```
SELECT [TOP N] column_1, column_2, ...  
FROM table_name;
```

- Examples

```
SELECT TOP 100 phot_mean_g_mag, bp_rp, parallax  
FROM gaiadr3.gaia_source;
```

```
SELECT TOP 100 *  
FROM gaiadr3.gaia_source;
```

2. Basic query structure

WHERE statement

- Syntax

```
SELECT column_1, column_2, ...  
FROM table_name  
[WHERE condition];
```

- Example

```
SELECT ra, dec  
FROM gaiadr3.gaia_source  
WHERE parallax > 20 AND duplicated_source = 'false';
```

2. Basic query structure

ORDER BY statement

- Syntax

```
SELECT column_1, column_2, ...  
FROM table_name  
[ORDER BY column [ASC/DESC]];
```

- Example (note that the orders of these commands do matter!)

```
SELECT TOP 200 ra, dec  
FROM gaiadr3.gaia_source  
WHERE parallax > 20  
ORDER BY ra;
```

2. Basic query structure

Operators & Comments

- Arithmetic operators

+ -- add
- -- subtract
* -- multiply
/ -- divide
% -- modulo

Comparison operators

= -- equal to
> -- greater than
< -- less than
>= -- greater than or equal to
<= -- less than or equal to
<> -- not equal to
IS NULL -- True if null
IS NOT NULL -- True if not null

- Logical operators

AND -- TRUE if both conditions are TRUE
OR -- TRUE if either conditions is TRUE
NOT -- TRUE if conditions is NOT TRUE

2. Basic query structure

Table alias

- Syntax

```
SELECT [tab_alias.]column_1, [tab_alias.]column_2, ...  
FROM table_name [AS tab_alias];
```

- Example

```
SELECT g.ra, g.dec  
FROM gaiadr3.gaia_source AS g  
WHERE g.parallax > 20 AND g.duplicated_source = 'false';
```

2. Basic query structure

Column alias

- Syntax

```
SELECT tab.column [AS new_column]  
FROM table_name;
```

- Example

```
SELECT g.ra, g.dec, g.phot_g_mean_mag AS gmag  
FROM gaiadr3.gaia_source AS g  
WHERE g.parallax > 20 AND g.duplicated_source = 'false';
```

2. Basic query structure

GROUP BY & Aggregation functions

- Group By: groups rows that have the same values into summary rows
- Aggregation functions: calculate statistics in the groups for specified column(s).

COUNT, MIN, MAX, SUM, AVG

Star Catalog		
Unique ID	Spectral Type	Teff
0	B	24983
1	B	20568
2	A	8988
3	F	6252
4	F	6344
5	F	6010
6	G	5777
7	K	4120
8	K	4870
9	K	4982
10	K	4200

GROUP BY
Spectral Type

Star Catalog		
Unique ID	Spectral Type	Teff
0	B	24983
1	B	20568
2	A	8988
3	F	6252
4	F	6344
5	F	6010
6	G	5777
7	K	4120
8	K	4870
9	K	4982
10	K	4200

Aggregation

Grouped summary table			
Group	Count (*)	MIN(Teff)	MAX(Teff)
B	2	20568	24983
A	1	8988	8988
F	3	6010	6344
G	1	5777	5777
K	4	4120	4982

2. Basic query structure

GROUP BY & Aggregation functions

- Syntax

```
SELECT COUNT(*), MIN(column_name), MAX(column_name),  
        SUM(column_name), AVG(column_name), ...  
FROM table_name  
GROUP BY column_name;
```

- Example (note that the orders of these commands do matter!)

```
SELECT g.phot_g_n_obs,  
       COUNT(*) AS counts,  
       AVG(g.phot_g_mean_flux_over_error) AS avg_g_snr  
FROM gaiadr3.gaia_source AS g  
WHERE g.parallax > 20  
GROUP BY g.phot_g_n_obs  
ORDER BY g.phot_g_n_obs;
```

2. Basic query structure

Readability

- ADQL does not care about
 - Capitalisations
 - New lines
 - Indentation
- But it would good to keep some conventions!

3. Common ADQL functions

Mathematical functions

- Syntax

`ABS(x)` -- returns the absolute value of x.

`EXP(x)` -- returns e raised to the power of x.

`LOG10(x)` -- returns the base 10 logarithm of x.

`LOG(x)` -- returns the base e logarithm of x.

`SQRT(x)` -- returns the square root of x.

`POWER(x,y)` -- returns the value of x raised to the power of y.

`MOD(x,y)` -- returns the remainder of x/y.

`ROUND(x,n)` -- round x to n decimal places.

`CEILING(x)` -- returns the smallest integer that is not less than x.

`FLOOR(x)` -- returns the largest integer that is not greater than x.

3. Common ADQL functions

Geometric functions

- CONTAINS checks if a point is wholly contained within a geometry
- Syntax

```
CONTAINS(point, circle/box/... ) -- returns 1 or 0
```

- Example

```
SELECT source_id  
FROM gaiadr3.gaia_source  
WHERE 1 = CONTAINS(POINT(ra, dec), CIRCLE(19, 56, 0.1));
```

3. Common ADQL functions

Geometric functions

- DISTANCE computes the arc length along a great circle between two points and returns a numeric value expression in degrees

- Syntax

```
DISTANCE(POINT(x1, y1), POINT(x2, y2))  
DISTANCE(x1, y1, x2, y2)
```

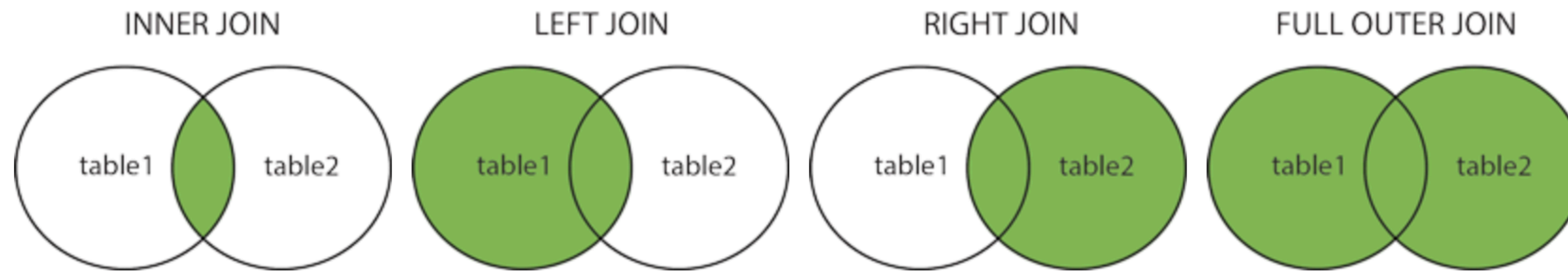
- Example

```
SELECT source_id  
FROM gaiadr3.gaia_source  
WHERE DISTANCE(POINT(ra, dec), POINT(19, 56)) < 0.1;
```

4. Joins and subqueries

Joins

- Inner join / left join / right join / full outer join (based on matching a unique identifier)



- Cartesian join (based on matching some conditions)

4. Joins and subqueries

Inner Join

- Inner join syntax

```
SELECT t1.col1, t1.col2, ..., t2.col1, t2.col2, ...  
FROM table1 AS t1  
JOIN table2 AS t2  
ON t1.unique_id = t2.unique_id -- matching a unique identifier
```

- Example

```
SELECT dr2.source_id, x.*  
FROM gaiadr2.gaia_source AS dr2  
JOIN gaiadr3.dr2_neighbourhood AS xmatch  
ON dr2.source_id = xmatch.dr2_source_id  
WHERE dr2.phot_g_mean_mag < 5
```

Kepler Input Catalog		Gaia Source		
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739	1001300000000	294	
7334352	5924	1001400000000	387	7334353
7334353	5827	1001500000000	132	

Inner Join based on KIC				
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334353	5827	1001400000000	387	7334353

4. Joins and subqueries

Left Join

- Left join syntax

```
SELECT t1.col1, t1.col2, ..., t2.col1, t2.col2, ...  
FROM table1 AS t1  
LEFT JOIN table2 AS t2  
ON t1.unique_id = t2.unique_id -- matching a unique identifier
```

Kepler Input Catalog		Gaia Source		
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739	1001300000000	294	
7334352	5924	1001400000000	387	7334353
7334353	5827	1001500000000	132	

Left Join based on KIC				
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739			
7334352	5924			
7334353	5827	1001400000000	387	7334353

4. Joins and subqueries

Right Join

- Right join syntax

```
SELECT t1.col1, t1.col2, ..., t2.col1, t2.col2, ...  
FROM table1 AS t1  
RIGHT JOIN table2 AS t2  
ON t1.unique_id = t2.unique_id -- matching a unique identifier
```

Kepler Input Catalog		Gaia Source		
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739	1001300000000	294	
7334352	5924	1001400000000	387	7334353
7334353	5827	1001500000000	132	

Right Join based on KIC				
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
		1001300000000	294	
7334353	5827	1001400000000	387	7334353
		1001500000000	132	

4. Joins and subqueries

Full Outer Join

- Full outer join syntax

```
SELECT t1.col1, t1.col2, ..., t2.col1, t2.col2, ...  
FROM table1 AS t1  
FULL OUTER JOIN table2 AS t2  
ON t1.unique_id = t2.unique_id -- matching a unique identifier
```

Kepler Input Catalog		Gaia Source		
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739	1001300000000	294	
7334352	5924	1001400000000	387	7334353
7334353	5827	1001500000000	132	

Full Outer Join based on KIC				
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334351	5739			
7334352	5924			
		1001300000000	294	
7334353	5827	1001400000000	387	7334353
		1001500000000	132	

4. Joins and subqueries

Certesian Join

- Cartesian join syntax

```
SELECT t1.col1, t1.col2, ..., t2.col1, t2.col2, ...  
FROM table1 AS t1  
JOIN table2 AS t2  
ON 1 = 1 -- matching a condition
```

Kepler Input Catalog	
KIC	Teff
7334350	5839
7334351	5739
7334352	5924
7334353	5827

Gaia Source		
source_id	n_visit	KIC
1001200000000	100	7334350
1001300000000	294	
1001400000000	387	7334353
1001500000000	132	

Cartesian Join				
KIC	Teff	source_id	n_visit	KIC
7334350	5839	1001200000000	100	7334350
7334350	5839	1001300000000	294	
7334350	5839	1001400000000	387	7334353
7334350	5839	1001500000000	132	
7334351	5739	1001200000000	100	7334350
7334351	5739	1001300000000	294	
7334351	5739	1001400000000	387	7334353
7334351	5739	1001500000000	132	
7334352	5924	1001200000000	100	7334350
7334352	5924	1001300000000	294	
7334352	5924	1001400000000	387	7334353
7334352	5924	1001500000000	132	
7334353	5827	1001200000000	100	7334350
7334353	5827	1001300000000	294	
7334353	5827	1001400000000	387	7334353
7334353	5827	1001500000000	132	

4. Joins and subqueries

Subqueries

- A subquery is a query nested inside a larger query.
- A subquery may occur in a `SELECT/FROM/WHERE` clause.
- See the example of proper-motion corrected spatial coordinates.

5. Cross match in ADQL

By unique IDs

- Cross match Gaia with 2mass.
- Tip: utilise pre-cross-matched tables.

```
SELECT TOP 1000
    gaia.source_id, gaia.bp_rp,
    tmass.designation, tmass.j_m
FROM gaiadr3.gaia_source AS gaia
JOIN gaiadr3.tmass_psc_xsc_best_neighbour AS xmatch
ON gaia.source_id = xmatch.source_id
JOIN external.tmass_xsc AS tmass
ON xmatch.original_ext_source_id = tmass.designation
```

5. Cross match in ADQL

By spatial coordinates

- Cross match two tables using a cone-search radius of 1 arcsecond.

```
SELECT TOP 1000
    gaia.source_id, gaia.parallax,
    hip.hip, hip.plx,
    DISTANCE(hip.ra, hip.de, gaia.ra, gaia.dec) AS ang_sep
FROM public.hipparcos AS hip
JOIN gaiadr3.gaia_source AS gaia
ON 1 = CONTAINS(
    POINT(hip.ra, hip.de),
    CIRCLE(gaia.ra, gaia.dec, 0.00028)
)
```

5. Cross match in ADQL

By proper-motion corrected spatial coordinates

- We need the following functions:

```
EPOCH_PROP_POS(ra/deg,  
               dec/deg,  
               parallax/mas,  
               pmra/(mas/yr),  
               pmdec/(mas/yr),  
               radial_velocity/(km/s),  
               ref_epoch/yr,  
               target_epoch/yr)
```

```
-- returns POINT(ra_target_epoch, dec_target_epoch).
```

```
-- only exists on Gaia Archive.
```

```
COORD1(point) -- returns the first coordinate of input point.
```

```
COORD2(point) -- returns the second coordinate of input point.
```

5. Cross match in ADQL

By proper-motion corrected spatial coordinates

- Cross match Gaia with galex for Pleiades stars (defined by ra, dec and proper motions), using a cone-search radius of 1 arcsecond.

```
SELECT subquery.*, galex.*
FROM (
  SELECT gaia.*,
         EPOCH_PROP_POS(gaia.ra, gaia.dec, gaia.parallax,
                        gaia.pmra, gaia.pmdec, gaia.radial_velocity,
                        gaia.ref_epoch, 2000) AS propagated_position_vector
  FROM gaiadr3.gaia_source AS gaia
  WHERE DISTANCE(56.75, 24.12, gaia.ra, gaia.dec) < 5.0
         AND SQRT(POWER(gaia.pmra - 20.5, 2) + POWER(gaia.pmdec + 45.5, 2)) < 6.0 -- Pleiades
  OFFSET 0 -- this increases query performance
) AS subquery
JOIN external.galex_ais AS galex
ON DISTANCE(COORD1(subquery.propagated_position_vector), -- ra_target_epoch
            COORD2(subquery.propagated_position_vector), -- dec_target_epoch
            galex.raj2000, galex.dej2000
            ) < 0.00028
```

6. Best practices and tips

For cross-matching on Gaia Archive

- User uploaded tables can be selected as user_username.table_name
 - e.g. user_yli02.k2_hermes
- Use gaiadr3.gaia_source_lite over gaiadr3.gaia_source
- Don't be lazy: don't select *
- Use indexed field and pre-computed fields over calculating new things on the fly
 - e.g. use bp_rp over phot_bp_mean_mag - phot_rp_mean_mag

7. Resources and further reading

- Gaia examples
 - More cross-match: <https://www.cosmos.esa.int/web/gaia-users/archive/combine-with-other-data>
 - Advanced usage and speed-up tips: <https://www.cosmos.esa.int/web/gaia-users/archive/writing-queries>
- ADQL documentation: <https://www.ivoa.net/documents/ADQL/20180112/PR-ADQL-2.1-20180112.html>
- TOPCAT

Exercises

- 1. Write an ADQL query that returns the nearest 100,000 Gaia sources
- 2. Write an ADQL query that returns all Gaia sources
 - Within 100 pc
 - With absolute G magnitude Gmag ($=\text{gmag}+5\log_{10}(\text{plx}/1000)+5$) as a standalone column
 - With Gmag between 4 and 6
 - With bp_rp between 0.8 and 1.2

Exercises

- 3. Write an ADQL query that
 - Retrieves all stars from exercise #1
 - It could be a subquery, or could be uploaded as a user table on Gaia Archive.
 - Cross-matches with hipparcos catalog based on spatial coordinates, with proper motion corrected.

1

```
SELECT TOP 100000 *  
FROM gaiadr3.gaia_source  
WHERE parallax IS NOT NULL  
ORDER BY parallax DESC
```

2

```
SELECT (g.phot_g_mean_mag + 5.*LOG10(g.parallax/1000.))+5.) AS Gmag, g.*  
FROM gaiadr3.gaia_source AS g  
WHERE parallax > 10.  
      AND ((g.phot_g_mean_mag + 5.*LOG10(g.parallax/1000.))+5.) > 4.)  
      AND ((g.phot_g_mean_mag + 5.*LOG10(g.parallax/1000.))+5.) < 6.)  
      AND (g.bp_rp > 0.8)  
      AND (g.bp_rp < 1.2)
```

3

```
SELECT DISTANCE(hip.ra, hip.de,  
                COORD1(gaia.propagated_position_vector),  
                COORD2(gaia.propagated_position_vector)) AS ang_sep,  
        hip.*, gaia.*  
FROM (SELECT TOP 100000 g.*,  
        EPOCH_PROP_POS(g.ra, g.dec, g.parallax, g.pmra, g.pmdec,  
                        g.radial_velocity, g.ref_epoch, 1991.25)  
        AS propagated_position_vector  
FROM gaiadr3.gaia_source AS g  
WHERE g.parallax IS NOT NULL  
ORDER BY g.parallax DESC  
OFFSET 0) AS gaia                CIRCLE(COORD1(gaia.propagated_position_vector),  
                        COORD2(gaia.propagated_position_vector),  
                        0.00028) )
```