NOTES ON PQL

Version 0.5

PQL

PARAMETRIZED QUERY LANGUAGE SYNTAX

BY

PAT DOWLER

DANIEL DURAND

LAURENT MICHEL

ALBERTO MICOL

FELIX STOEHR

Why PQL

Generalities

Why a Parametrized Query Language (PQL)? PQL is a very good complement to ADQL or the Astronomy Query Language designed for the Virtual Observatory as a common query language. PQL could simplify buy-in from non IVOA compliant sites in providing an IVOA standard which could be implemented simply. But essentially PQL has been designed as an alternative query mechanism for TAP services.

PQL has some advantages

- Easier parsing and implementation
- Easier Buy-In
- Easy to translate in ADQL if the service supports it
- Compatible with almost all existing form based service (with small modification)
- Extensible
- Easy to embed in a scripting language in order to support data mining experiment with large collection of images (data mining)
- Easier to remember than direct ADQL at the expense of simpler capabilities.

Why a Different PQL

Generalities

We would like to propose a different approach to the PQL protocol as stated in the document entitled "IVOA TABLE ACCESS PROTOCOL, PARAMETERIZED QUERY LANGUAGE". The protocol is actually fine as it is but we would like to propose a very different syntax than the one stated in the document. We think that the one proposed is actually counter-intuitive and, although quite logical for expert in GIS field, is very difficult to remember if it is used sporadically. It is also a very different syntax to what astronomers have been using for years on the web. There are already a lot of "archive sites" which are available to a semi common form language. We believe that any deviation to what is already widely used might reduced dramatically the buy in effect.

The basics syntax

We would like to propose a parameter language based on very simple construct using pre-defined logical and easy to remember operators. The general syntax is simply:

column_name <==> operator <==> value

or

utype <==> operator <==> value

The <column name> value could be the actual column name in the target table or any valid <utype>. Please note that if <utype> usage is desired, one cannot mixed <utype> and column name in the same query. When <utype> is used, the table name is usually not required if unique in a given model like ObsCore.

Here the list of the main supported operators:

The operators

FUNCTION	EXAMPLE	EXPLANATION	NOTES
..	col = 3.0..75.0	col is between 3.0 and 75.0	
,	col = a, b, d, e	col could be equal to a, b, d or e. This is an or operator	
<,<=	col = <=1.0e-2	col is less or equal to 1.0e-2	Has to be url encoded
>,>=	col = >10	col is greater than 10	Has to be url encoded
!	col = !(a,b,d,e)	col is NOT any of a,b,d and e	Use () for distributing the operator
case	col = case(toTo.fits)	col match TOTO.FITS or toto.fits	Case sensitive operator
?	col=toto?.fits		standard dummy
*	col = *fits	col will match *.fits. Please note that an exact match don't need any function and can be written like col=*Her or col=14	
ESCAPE character			
\	col=\^	Escape operator	

So it is then implicit that all the conditions are ORed together. So a construct like:

a=foo&a=blah

is valid and will select a with values with a = foo or a=blah, equivalent to the list operator.

As an alternative, one of us would prefer using operator like “gt” instead of “>”, etc... SO a standard PQL statement would be like:

cgi?table=TableName(col1,col2)&position=inCircle("M33")&colName=gt(3.45)

Other operators

OPERATOR	NOTE	SYNTAX
select	list of <column name> / <utype>. (all is the default is the select is not specified)	select=* (select all) select= col1, col2, etc... (select a list) select=ivoa:ObsCore (select the mandatory columns from a specific model name)
table	list of table names. I guess we could assume that ivoa.ObsCore is the default is none specified. So join are then possible if multiple columns/ UTYPE are listed. Self-join not allowed. Don’t have to be specified when utype are used.	table=blah (generic table) table=ivoa.ObsCore (the ObsCore standard table)
OR		
table=table_name(list_of_columns)	merging table and select	

Some examples:

select= (col1, col2, col3, col4)&table=my_table&col4=value1..value2&col2=>=2.034&col1=<=-1.0E-03

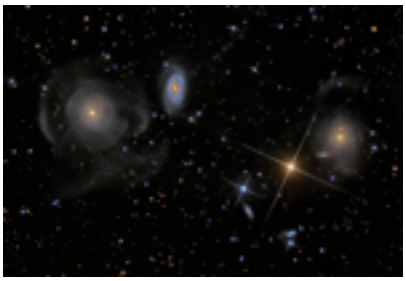
will do the following:

select col1, col2, col3 and col4 from my_table where col1<=-1.0E-3 and col2>2.034 and col4 is between value1 and value2

select=*&table=ivoa.ObsCore& s_region=points(<ICRS>, 12., 34.)

select all the mandatory columns from the ivoa ObsCore model and search all entries which include the

point centered at ra=12. and dec=34.


Positional Parameters

For positional parameters, I think it is better to use the STC-S syntax which could be easily mapped into already existing tables. Please note that not all STC-S functions has to be supported. One server could return a good approximation if geometric index are not available.

We identified three flavors

- Queries using VO tables and columns names
- Queries using utypes
- Queries using some magic (for non VO compliant sites/values)

Case 1: Queries using VO tables and columns names

table=ivoa.ObsCore&s_region=circle(<ICRS>, 12, 34.) (ivoa obscore)

table=ivoa.ObsCore&s_region=circle(resolve(m31))

Case 2: Queries using UTYPES

Although quite ugly to read, the query by UTYPE is

&obscore:Char.SpatialAxis.Coverage.Support.AreaChar.SpatialAxis.Coverage.Support.Area = circle(<ICRS>,12, 34.) (using utype?, table not required)

Case 3: Queries for non VO compliant implementation

This syntax for querying positional parameters is available mostly for services which are not VO compliants. Then it is up to the server to map the desired coordinates. For this purpose, we are using the magic directive POS (Position Orchestrated by Server). Vo compliants services will have no problems supporting this syntax.

pos = point(<ICRS>, ra, dec) => Define a starting coordinate for the search. Default is ICRS.

pos = (resolve(object_name)) is a short cut and include an implicit resolver and ICRS

pos = box(<ICRS>, a1, a2,b1, b2)

pos = polygon(<ICRS>, etc...)

and in more general term

pos = region(stc_query syntax)

Additional information

Positional Parameters

The values of RA and DEC in degrees or in sexadecimal. Please note that when specified together, the sign act as a separator between RA and DEC and if DEC is positive, the + sign will have to be url encoded, i.e %2b.

For using different coordinate systems like Equatorial and Galactic, one has to specify it like

pos = polygon(GAL, etc...)

s_region = polygon(GAL, etc...) for VO

Energy parameters

The main problem in specifying the energy parameters is how to encode the desired units. We suggest we could eventually support those units values as suffixes to the numbers. To be discuss but we feel that, for the first PQL implementation, units should not be specified and it is up to the "client" to adapt his/her searches to the units provided by the data provider otherwise it will have to be too complicated. The units problem is way difficult and should be addressed in a large scope than this document.

Frequency Units	"Hz","kHz","MHz","GHz"
Ev units	"eV","keV","MeV","GeV"

Wavelength Units	“m”,”cm”,”mm”,”um”,”nm”

Any values could be followed with one of these above units. Conversion will then occurs on the server side. The default unit is the one offered by the server and the user will have to read the documentation of a given site.

Time Parameters

time = value

where value is specified as normal FITS time value i.e. yyy-mm-ddThh:mm:ss.s

or in MJD which is the default

Some magic values and function as useful like

time = today() or now()???

Need discussion here....

Other reserved values (in addition to PQL)

Although not part of the data model but part of DALI, those values are useful when querying vast ensemble and could be generically supported, specially for non VO compliant sites. For example:

MAXREC = 40 => Setup the maximum number of results (DALI)

REQUEST (DALI) (the way to get the documentation is REQUEST=capabilities?

dry-only -> is simply specifies as MAXREC=0

Examples

Example 1:

Use PQL to query my table named ***my_table*** which is an object catalogue which has multiple coordinate columns, one per colour, i.e. ra_b, dec_b, ra_v, dec_b, ra_u, dec_u.

select=*&table=my_table& pos=box(<ICRS>, ra_1, ra_2, dec_1, dec_2)

in that case the implementation has to select which pair of ra,dec will correspond to the pos of a given object

Now if the table do have the standard support for point and reqion query, one could write the same example as:

select=*&table=my_table& s_region=box(<ICRS>, ra_1, ra_2, dec_1, dec_2.)

Example 2:

Using the PQL to query a table similar to the ivoa.ObsCore data using the column names mapped magically by the server, i.e. assume the Obscore table does exist but there is no TAP service nor STC-S support.

select=all&pos=resolve(M31)&calib_level=2&dataproduct_type=image

Example 3:

Using PQL to query the ivoa.ObsCore data using the utypes:

obscore:DataID.collection=JCMT

obscore:Obs.dataProductType=spectrum

http://www.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/tap/sync?REQUEST=doQuery&LANG=PQL&MAXREC=10&obscore:DataID.collection=JCMT&obscore:Obs.dataProductType=spectrum

REFERENCES

Please see older PQL like older protocol: (http://cds.u-strasbg.fr/doc/asu.html)

Some of the protocol was inspired by:
http://code.google.com/apis/analytics/docs/gdata/gdataReferenceDataFeed.html

The Original PQL document by Doug Tody

http://www.ivoa.net/internal/IVOA/TableAccess/PQL-0.2-20090520.pdf