THE INTERNATIONAL VIRTUAL OBSERVATORY ALLIANCE

Recommendations for a Table Access Protocol

Ray Plante,
Tamas Budavari, Gretchen Greene, John Goode,
Tom McGlynn, Maria Nieto-Santistaban,
Alex Szalay, Roy Williams

Some Lessons Learned

- Experience with ADQL/x
  - Motivation behind ADQL/x:
    - Query Transformation is commonly necessary
      - Few databases are 100% compliant with the SQL standard.
        - » Transform to local SQL dialect
      - Semantic filtering possible (transforming metadata).
      - Easier to adapt to non-relational databases (e.g. XML database)
    - *Supposition:* A pre-parsed form on the wire makes transformations easier to implement
  - Experience:
    - Shifts parsing problem to the client –
    - Minor transformations can often be handled via simple SQL string manipulations
    - More careful adherence to SQL92 would eliminate most common difference between native SQLs (TOP, functions)
    - The emergence of parser/conversion tools make choice of wire format less important
  - Lesson:
    - Allow string-based SQL on wire
    - Stick closer to standard SQL syntax

Some Lessons Learned

- Regions and Cross-match
  - "Magical" function definitions
    - (originally) functions did not specify what columns should be used in the calculation
    - Cross-matching required certain, unspecified columns to appear in the response.
  - Consistency required in use of cross-match
    - Implementation must be well defined
    - Users will want to use same implementation at all sites being matched
  - Import/export of XML table data is costly
    - Can we take advantage of fact that multiple surveys are on the same server?
- Implementing a SkyNode is hard
  - Can the simplest implementation send simple SQL to a database without tranformation?

How might we benefit from these lessons in a Table Access Protocol?

TAP taps into a “Table Set”

- A collection of tables accessible via a single access URL
  - One or more tables
  - Join between tables is, in principle, is possible
    - E.g. within a single DBMS or logical equivalent
- Table typically logically related
  - e.g. A CDS “catalog”, all tables of SDSS DR4
- Collections can be aggregated for performance purposes
  - e.g. all CDS catalogs, SDSS+2MASS+FIRST
- Client can take advantage of tables being co-located
  - Local joins, XMatches

Character of a TAP

- Carrier protocol
  - GET, POST, or SOAP supportable
  - Some advanced queries may not be supportable with GET, POST
- Operations
  - Search
    - Query
    - Query format used
      - Native SQL, ADQL/s, ADQL/x, …
    - Output format desired
    - Top/Offset selections
    - Disposition – what to do with results
      - Return to caller synchronously, save in store for later retrieval
  - Upload: returns a name and longevity
  - getCapabilities: what QL features are supported
  - TableSet: describe tables, columns available (as queryable tables?)
- Notice that Query Language does not require…
  - TOP/OFFSET
  - SELECT INTO
  - UPLOAD

Approach to a Standard Query Language

- Maximize adherence to SQL92
  - Enable minimal transformation to native SQL
  - Makes string format convenient on wire
    - XML format may defined (perhaps separately) if useful for an implementation
- Allow features to be grouped into sets for graduated support
  - Core language feature set
  - Protocols (e.g. TAP) indicate which sets are considered required, which optional
  - Protocol capability metadata declare support for which optional feature sets

Standard Query Language

Basic Syntax

- Core Syntax

```
SELECT id, ra, dec, jmag FROM objcat
WHERE jmag < 18.0
```

  - No SELECT INTO, no data manipulation
  - Optional: aliases, standard schema names

```
SELECT p.id, p.ra, p.dec, m.jmag
FROM survey.objcat p, survey.magnitudes m
WHERE m.jmag < 18.0 AND p.id=m.id
```

  - Core operators:
    - AND, OR; >, <, >=, <=, <>, …
    - BETWEEN, NOT BETWEEN, LIKE (string comparison)
  - Standard Function syntax supported
    - Allow support for implementation-specific functions
    - Core set of functions: abs(), pow(), ?…
- Extended Function Sets
  - Group other commonly supported functions into sets
    - Trig, aggregators, …
    - Service description can indicate support for whole groups
- Table Joins
  - Implicit joins (as above) part of core syntax
  - Explicit joins (INNER, OUTER?,…): extra-core
- ORDERBY (extra-core)

Standard Query Language

Regions

- Enable explicit declaration of position types

```
Position(p.ra, p.dec)
Position(p.obsra, p.obsdec)
Position(p.x, p.y, p.z)
Position()  – implementation determines default position*
Position(p.ra-0.05, p.dec-0.05)
```

*important for optimization

- Region testing functions return boolean

```
RegionContains( <region-spec>[, <position-spec>} )
RegionContains('CIRCLE ICRS 120.0 30.0 1.0', Position(p.ra, p.dec))
RegionContains('CIRCLE ICRS 120.0 30.0 1.0')
          – implementation determines default position*
```

- Advantages
  - Eliminate magic: positions explicitly specified
  - Allow functions implemented either as
    - stored procedures, or
    - with simple string substitutions

User-supplied Tables

- UPLOAD, SELECT INTO *not part of standard language*
  - Protocol handles this separately
- Convention for naming user-supplied tables
  - Schema name: “@upload”

```
SELECT u.objid, u.flux, b.ra, b.dec,
FROM   "@upload.primary" u, sdss.photoprimary b
WHERE  u.objid=b.objid
     AND RegionContains('CIRCLE ICRS 10. 40. 2')
```

  - @ avoids collision with real schema names
  - Requires quotes to escape SQL parsing issues
  - Upload process assigns table name

Standard Query Language

XMatch syntax

- XMatch: a table described as a function in the FROM clause

```
SELECT u.objid, u.r, u.ra, u.dec, m.m_ra, m.m_dec,
FROM   "@upload.primary" u, sdss.photoprimary b, xChiSq(b,u) m
WHERE  b.r < u.g AND m.m_chisq <10
       AND Contains('CIRCLE ICRS 10. 40. 2')
```

- Application of function produces a query-able table
- An XMatch function definition includes
  - Definition of input values
    - *Should* allow user to specify what position values in record to use!

      ```
      xChiSq(b.ra, b.dec, u.x, u.y, u.z)
      xChiSq(b.ra, b.dec, u)
      xChiSq(b.ra-0.05, b.dec-0.05, u)
      ```

    - *May* provide syntax that allows implementation to decide for optimization
  - Definition of schema of generated table
  - Formal definition of calculation that produces those table values
- Advantages
  - Extendable to any sort of cross-match
  - Allows client to control exactly what is returned in result via std. SQL
  - Eliminates "magic"
- Disadvantage: departure from standard SQL
  - Alternative: a suite of functions usable in SELECT & WHERE; SQL-compliant!
- Note: syntax is part of language—not the specific cross-match functions

Registering a TAP

- Describing underlying collections
  - Simple single Table Set: described as part of TAP service record
  - Table Sets that access multiple surveys
    - Register collections separately
    - Refer to collections by identifier
- TAP Capabilities:
  - Query languages supported
    - Native: vendor & version, notion of what's (not) allowed
    - ADQL: sets of language features supported
  - Underlying protocols supported
  - Return formats supported
  - Dispositions supported (asynchronous mechanisms included)

A SkyPortal using TAP

- Portal uses capabilities to make best use of tables
- Nominally, SkyNode = Core+Regions+uploads+XMatch
  - Some TAP Implemenations may not be available for cross-match
  - A smart portal may work around limitations
  - Portal searches for TAP services with capabilities it requires
- ExecutePlan not needed!
  - TAP's Disposition parameter allows portal to tell TAP service exactly what to do with results.
  - Portal can orchestrate other complex query workflows
    - Not restricted to current single chain
- Issue: how to calculate query costs
  - Part of TAP? Advanced TAP?