CQL 2.0

# Use Cases for CQL 2.0

## Associated Object Retrieval

From: Imaging, TBPT, many others

Overview: Users need to be able to return a target object with one or more of its associations populated. Potentially also the association’s associations populated as well.

Example: A researcher would like to retrieve all Genes with a symbol like ‘BRCA%’ and then get the chromosome number affected. Currently, the researcher would have to perform two queries: The first to get all Genes with symbol like ‘BRCA%’, then another to get all Chromosomes who have an association to a Gene with one of the IDs of the genes from the first query. It’s cumbersome, slow, and causes unneeded network traffic. Additionally, such a two-step query is impossible (does not conform to the domain model) if the association from Gene to Chromosome is not bi-directional.

## Temporal Queries

From: TBPT

Overview: Expression of query attributes in terms of ‘age’.

Example: A researcher would like to find all Samples in a data service that are 30 days old or newer. While one could formulate such a query presently by doing something like “current day – 30”, the query cannot be saved for later re-use.

## Aggregate Queries

From: TBPT, few others

Overview: Return things like min, max, average values. Also use min and max in the query itself.

Example: A researcher would like to know the max white blood cell count from a set of blood samples defined by some query.

Example2: A researcher would like to return the blood sample with the fewest white blood cells.

## Strong Data Types in Query attributes

From: Everybody

Overview: CQL Attribute has a ‘value’ attribute, which is presently just a string. Users don’t know how to format dates, times, etc.

Example: Overview says it all. Adding this feature removes an ambiguity and isolates CQL formatting from the backend data source’s expected formatting.

## Complex Attribute Types

From: TBPT

Overview: Ability to query over ‘attributes’ which are actually a conglomeration of attributes.

Example: An Identifier field may be of the format “ssn:firstName:lastName” or similar. A researcher wants all Patients with identifier.firstName = “Foo”.

## Returning attributes from multiple objects

From: TBPT

Overview: Ability to return an attribute value from two (or more) different objects.

Example: A researcher would like all white blood cell counts from blood specimens, as well as dates of hospital visits for a set of patients defined by a query.

## Returning subclasses of domain objects

From: caTissue, others

Overview: Users would like to retrieve a class and its subclasses when queried

Example: A researcher queries for the base class ‘Specimen’ and expects to get back both BloodSpecimen eand TissueSpecimen instances.

# Solutions

## Associated Object Retrieval

Thoughts: This is very common request, so it deserves high priority. I propose adding an optional element to the root of a CQL query (like Query Modifiers are now) to specify which associations are to be returned. The association population should be configurable to either populate ALL associations up to a certain number of levels, or a named association (or multiple associations). In the case of named associations, the query developer should be able to specify the names of sub-associations to populate as well, in a recursive fashion.

Proposal: The schema AssociationPopulationSpec.xsd fulfills the requirements. It allows for a choice of depth-based population or named association population. The naming is specified recursively, and the depth based population is a simple integer value. The schema presently allows for a flag to be set indicating “infinite” depth population, the usefulness of which needs to be evaluated.

To-do:

I. Evaluate the usefulness and practicality of implementation of the ‘infinite depth’ flag

II. Naming of elements and types in the schema needs some work

## Temporal Queries

Thoughts: In the TBPT case, this might be a modeling issue. Things like “age” should never be stored in a database, since they change as time progresses. From a query perspective, we would need a way to make a query “relative to” some other value. This gets into the area of joins, which CQL doesn’t really do. In this case, the value is relative to today’s date, so it’s a known value and not really a join but a value replacement on the server side.

To-do:

I. Evaluate some TBPT models to see if anybody actually stores “age” values.

II. Develop a specialized query type for temporal queries

A. Might be dependent on the strongly typed queries values request

## Aggregate Queries

Thoughts: In the case of returning min and max, it’s easy to just add these to the query modifier. Average would be difficult in the case of non-numeric values, but probably not impossible from an implementation standpoint. Including min, max, and average in the query itself is more difficult. CQL Attributes could add some predicates.

Proposal: The schema Aggregations.xsd defines a simple type which contains the supported aggregation operations. Currently it contains COUNT, MIN, and MAX, and has AVERAGE commented out until we can evaluate its usefulness. This type should be referenced from the QueryModifier type.

To-do:

I. Add aggregates to query modifiers

II. Evaluate what to do for non-numeric values (Strings and Dates)

III. Develop a new CQL Attribute / Predicate

A. Related to the strong data typing in attributes

IV. What do we do with attributes in queries when there is more than one data instance which fits the aggregation? (eg. 5 samples, two have the same white blood cell count)

## Strong Data Types in Query attributes

Thoughts: This may require a much more complex XSD type to make correct. CQL Attribute could be a base type, with just the name of the attribute. Then Unary predicates get their own sub type (IS NULL, IS NOT NULL, MAX, MIN, AVERAGE), and Binary predicates get another. The data types may also have to be an XSD choice, since an XSD union ends up as just a String when Axis generates beans from it.

To-do:

I. Draft a new CQL attribute schema type, potentially a few versions

II. Evaluate the added complexity of the proposed attribute type

A. Ideally, all the new information I need could be part of the same element

## Complex Attribute Types

Thoughts: ‘Complex’ attributes present a tremendous challenge to parse correctly, since they’re something in-between a full blown association and a true attribute. In general, this is a modeling problem, where the attribute should be broken out to its own type with individual attributes attached, and an association to the parent object formed. There’s no good way to express a complex attribute in XML; it would probably appear as just a String. If unavoidable, the query developer might be able to use wild cards to emulate certain queries, but only depending on the syntax of the attribute. For example, an attribute of the form ‘ssn:firstName:lastName’ could be queried for first name only by using ‘%:FooFirstName:%’.

To-do:

I. How prevalent in caBIG are these kind of modeling issues?

## Returning attributes from multiple objects

Thoughts: I have no idea how this might work in a purely hierarchical query language. Only one target may be specified, so there’s no way to define the additional attribute(s) to return. The information required could potentially be obtained by using association population, at the expense of returning data the client is not interested in.

To-do:

I. Will association population handle this case?

II. How would one express a tuple of results from different objects?

III. How are those tuples built? Cross join, left join?

## Returning subclasses of domain objects

Thoughts: (From Scott):

*Consider the example of a client looking for instances of*

*caBIO's Gene class where a client discovers multiple data services*

*providing data of that class, and issues a federated query to all of*

*them. It's quite possible that many of the services are exposing domain*

*models wherein they have extended/subclasses the Gene class. If we*

*returned subclasses for queries for Gene, the client would receive all*

*these different subclasses as well. The problem with this is the client*

*may not know how to process these arbitrary subclasses. To process the*

*results, the client needs to either deserialize them into their Object*

*representations, or deal with the XML directly. If the client intended*

*to deserialize them, it would actually have to have all the subclass*

*Classes on the classpath, in addition to the caBIO model. This would be*

*a problem, as arbitrary new subclasses could be introduced to the grid*

*at any time, and this client would have to have all of them. The other*

*option is to process the XML directly. The problem with this is that*

*the subclasses may actually change the XML representation of the Gene*

*class in order to serialize themselves. That is, the client can't just*

*assume that because it knows how to extract the "symbol" attribute from*

*the Gene XML, that is can do it the same way for any of Gene's*

*subclass's XML representation. This is why we only return specifically*

*what the client asks for.*

Maybe we can provide some control over subclass processing. Should subclasses of targets be returned? All / some / named? If we implement association retrieval, does this control extend to associations and their subclasses as well? Does it make more sense to embed the subclass handling in the data service itself and publish it as metadata?

To‑do:

I. Evaluate subclass handling for target data types in the general (all) case

II. Evaluate user control over subclass handling (some kind of XSD spec to be attached to each query)

III. Evaluate service-defined subclass processing