Structured Query in the EON Guideline Model

# Introduction

“Structured Query” is a new addition to the EON Guideline Model. It allows the construction of relatively complex queries that hitherto had to be formulated as PAL queries. This document tries to give a user some idea about what can be expressed as “structured queries” and examples of how to construct such queries.

# Basic Operations

A Structured Query is patterned after SQL queries of the form

Select (aggregation operator) (column) from table where …

It has the form

Select (*aggregation operator*)(*attribute*) from *type* where (*Filter*)\*

where the mandatory *type* is a Protégé class, the optional *attribute* is a slot of the class, the optional *aggregation operator* is one of {*average, count, maximum, minimum, most\_recent*}, and (*Filter*)\* is zero or more instances of the **Filter** class. In Figure 1, **Adverse\_Reaction** is the type and the where restrictions are ‘domain term is a subclass of Angioedema’ and ‘substance is subclass of ARB.’

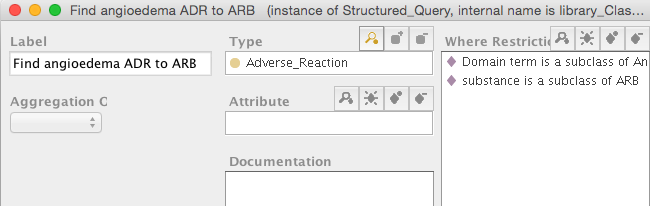


Figure . An example of "Structured Query"

The execution engine resolves a “Structured Query” by

1. Finding all instances of *type*
2. Evaluating the instances against the *where restrictions*
3. If the aggregation operator is *most\_recent*, finding the most recent instance of the instances that satisfy the *where* *restrictions*
   1. if *attribute* is not null, returning the attribute value of most recent instance
   2. if *attribute* is null, returning the most recent instance
4. If the aggregation operator is one of {*average, maximum, minimum*}, the *attribute* must not empty and must have numeric values. In this case the execution engine applies the aggregation operator to the numeric attribute values and returns the result.
5. If the aggregation operator is *count*, returning the count of instances that satisfy the *where restrictions* (in case there is no attribute) or the count of attribute values of the instances that satisfy the *where restrictions*
6. If there is no aggregation operator, returning the set of instances that satisfy the *where restrictions* (if there is no attribute) or the set of attribute values of the instances that satisfy the *where restrictions*.

# Filter

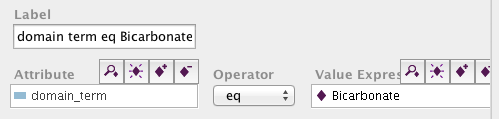
A filter associated with a query is evaluated against an instance of the type in the query (e.g., Medication). If the evaluation results in *true* then the instance is passed to the next filter. If an instance passes all filters of a query, then the instance either becomes the part of the query result or it is further processed according to the *attribute* and *aggregation operator* properties of the query. There are four types of filters.

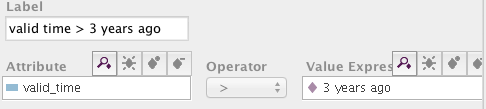
## Comparison\_Filter

Comparison filters have the form:

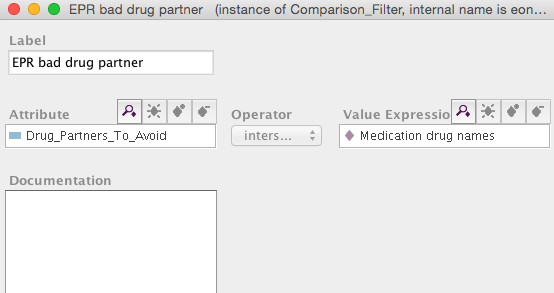
attribute operator value-expression

For example, when querying for instances of Note\_Entry or Numeric\_Entry, the comparison filters may be:

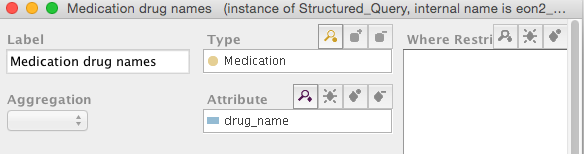




The *value expression* slot can be filled by any instance of the Expression class. In particular, it may be another Structured Query instance. So, if we want to find all instances of Drug Usage whose drug class name includes one that has a bad drug partner among current medications, we can construct a Structured Query with type = Drug\_Usage and whose where restriction is a Comparison Filter



where the *operator* is ‘intersection’ and the *value expression* is a the result of another Structured Query that get all drug names of current medications:



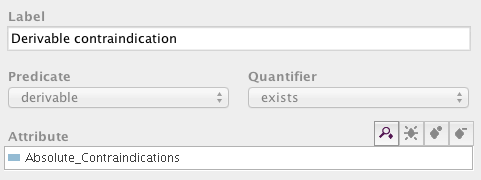
The comparison filter checks that a Drug Usage instance whose ‘Drug\_Partners\_To\_Avoid’ intersect the names current medications. The execution engine expands ‘Drug\_Partners\_To\_Avoid’ to all of its children when it performs the ‘intersection’ calculation.

## N\_aryFilter

An instance of N\_aryFilter is a Boolean combination of other Filters.

## Predicate\_Filter

An instance of *Predicate\_Filter* has three slots: *attribute*, *predicate* (always “derivable” for now) and *quantifier* (always “exists” for now). Currently the filter can check to see whether there exists a value of the *attribute* that satisfies the *predicate*. The use case for this filter is the need to check, for example, that one of the absolute contraindications of Drug Usage can be “derived” from patient data. ACE inhibitor has ‘Angioedema due to ADR to ACEi’ as an absolute contraindication. When the following filter is placed on instances of Drug Usage, and if the patient has angioedema due to ADR to ACEi, then the ACE inhibitor instance of Drug Usage will pass the filter.



## Join\_Filter

A *Join\_Filter* is designed to do a SQL-like join of two sets of instances. It implements a query of the form:

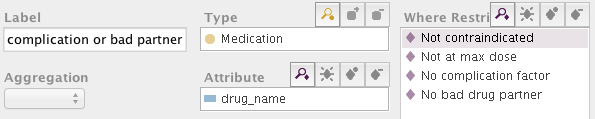
Select (*aggregation operator*)(*attribute*) from *type* t where [Query q and (t.attribute operator q.attribute\_to\_compare)\*]

A *Join\_Filter* implements

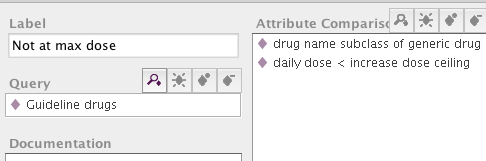
[Query q and (t.attribute operator q.attribute\_to\_compare)\*]

It has two slots *query* and *attribute comparisons*, that correspond to Query q an the set of join comparisons.

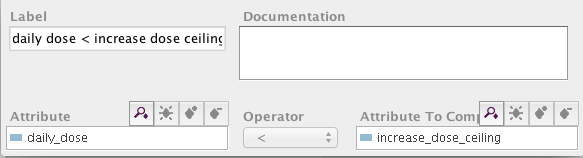
For example, to query for medications not at maximum dose, construct a query



where ‘Not at max dose’ is a join filter:



The filter get all instances of Guideline drugs and compare the attribute values of Medication instances with attribute values of Guideline drugs. For instance, ‘daily dose’ of Medication should be less than the ‘increase dose ceiling’ of a Guideline drug:



# Grammar

The Structured Query uses Protégé to implement the following query grammar:

Notation:

Word\_In\_Regular\_Font represents a non-terminal node defined by a production

*Italic\_Word* represents a symbol

“|” represents “or” (i.e., possible alternative)

(..) represents 0 or 1 occurrence (i.e., optional)

(..)\* represents 0 or more occurrences (i.e., optional, may have multiple entries)

[..] represents 1 occurrence (required, and only one entry is allowed)

[..]\* represents 1 or more occurrences (i.e., required, multiple entries are allowed)

Structured\_Query :

**Select** (*most\_recent*) (attribute) **from** [type] (where\_restriction)\* |

**Select** *count* (attribute) **from** [type] (where\_restriction)\* |

**Select** [numeric\_aggregator\_operator] [attribute] **from** [type] (where\_restriction)\*

type: a Protégé class that is a subclass of Guideline\_Model\_Entity, EPR\_Entity, or

Canonical\_Term\_Metaclass

numeric\_aggregator\_operator:

*average* | *maximum* | *minimum*

attribute: a Protégé slot that is a template slot of the class *type*

where\_restriction:

Comparison\_Filter | N\_aryFilter | Predicate\_Filter | Join\_Filter

Comparison\_Filter:

(attribute) [comparison\_operator] [value\_expression]

comparison\_operator:

*> | < | <= | >= | = | eq | neq | member\_of | subclass\_of | superclass\_of | intersection*

N\_aryFilter:

[logical\_operator] [where\_restriction]+

logical\_operator:

*AND* | *OR* | NOT

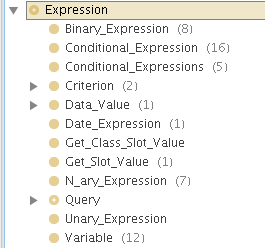
Predicate\_Filter:

(attribute) quantifier predicate

quantifier: *exists* | *forall*

predicate: *derivable*

value\_expression: an instance of a subclass of Expression



Join\_Filter: [Query] [attribute\_comparison]\*

Query: Structured\_Query | PAL\_Query

attribute\_comparison : [attribute] [operator][attribute\_to\_compare]

See 3.4

operator :

> | < | <= | >= | = | eq | neq | member\_of | subclass\_of | superclass\_of | intersection | not\_subclass\_of | not\_member\_of

# Patterns

This section describes some common patterns of making structured queries.

## Basic Queries

**Pattern:**

**Select** \* **from** [type] (attribute comparison\_operator value\_expression)

**Examples:**

1. select \* from Medication (daily\_dose > 20)
2. select drug\_name from Medication (daily\_dose >20)(drug\_name subclass\_of ACE\_Inhibitors)

Comments:

* “attribute” should be a template slot of the class that’s the focus of the query. In Example 2, both “drug\_name” and “daily\_dose” are slots of the Medication class.
* Multiple “where\_restriction” in the selection frame are ANDed together. Therefore, in Example 2, it’s not necessary to use n-ary filter to AND together the two comparison filters (one involving daily\_dose and the second involving drug\_name)