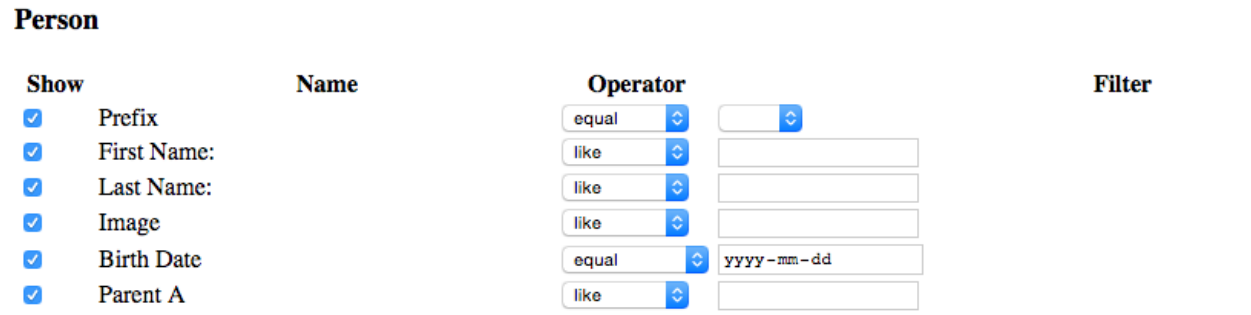
Query By Example-(QBE) Search Form Construction

This is part of the query wizard project, but will also be used independently as a standalone component. The goal is to start with a JavaScript object, which represents the meta-data about a “class”, and produces an HTML form which helps the user specify a structured search query, for the data in that class. Remember that a “class” roughly corresponds to a database table, while “attributes” have the same relationship to the table’s columns.

An HTML example file, and the matching JavaScript object are provided with this document. The file “*class-query-form.html*” is an example of the expected result, and the file "*class-meta-data.json"* is the sample input data.

Here is a screenshot of the (unstyled) desired result.



The component’s responsibility is to produce an HTML *form* DOM element, when provided with an appropriate JavaScript object as the meta data. As a follow on to this project, we will create the ability for the component to convert the HTML form back into a different JavaScript object which represents the user’s selections. But for now, just construct the HTML DOM object for the *form*.

Description of the target form:

The form contains a hidden input for the CLASS.CLASS_ID, and an HTML *table* for layout. Each row of the table will correspond to one of the attributes (which is an array) in the JavaScript object. The table has four columns.

1. **Show :** A checkbox indicating whether this column will show, be included, in the resulting query display.
2. **Name :** The value of the “attribute” object's name property.
3. **Operator :** An HTML select with a choice of “operators", this varies by type and will be detailed below.
4. **Filter :** The HTML form *input* into which the user can enter filtering criteria, varies by type.

Description of the JavaScript Meta Data:

A JavaScript object, containing the following properties:

- **id** : A numeric id, this value is the primary key, and must be preserved throughout.
- **description** : A text description of the class, this is not involved in this project and can be ignored.
- **name** : Text, this is the name by which the user will recognize the class. It is displayed at the top.
- **package** : A containing object that can be ignored by this project.
- **privs** : An object which represents the current user’s privileges with respect to this class. Won’t be used.
- **attributes** : An array of objects, which form the basis for definition of the form’s fields.

Then each attribute, of the attributes array, will have a set of common properties, while some types will have additional specific properties:

- **id** : Numeric id for the attribute, this is used as the programmatic identifier.
- **label** : This is the text, or label, by which the user will recognize the attribute.
- **type** : A numeric code indicating the “type” of the attribute.
- **typeName** : Text value representing the type of the attribute.
- **pos** : A number indicating the position value, used as the sort order for the attribute collection.

There are additional properties in the attribute object, but they won't be used for this application.

When the "Filter" column of the form is constructed, the following HTML constructs should be used for the different attribute types.

1. **Pick List** attribute objects contain a "list" array, each item in the list is a numeric key, and a text value. In the form this attribute type will be represented by an HTML select, the numeric key is the option value, and the text is the option's content.
2. **Date** attributes should be represented as an HTML input element of type date.
3. **Number** attributes should be represented as an HTML input element of type number.
4. All other types can be represented as HTML input elements of type text.

Description of the Operator Lists:

The "operator" options set for the HTML select are defined by the attribute type.
PickList : will have equal, not equal, is null, is not null.
Text : like, equal, not equal, not like, is null, is not null.
Date : equal, not equal, less than, greater than, is null, is not null.
Number : equal, not equal, less than, greater than, is null, is not null.
Relationship : like, equal, not equal, not like, is null, is not null.

Important note! Please see the *class-query-form.html* file for exact examples on how the HTML should be constructed for the operator dropdowns. The option's value attribute does not match the display text and *includes significant whitespace*.

Additional notes:

In the class-query-form.html file you will see programmatically created HTML id, and name, attribute values. They are constructed according to the following patterns.

For attributes:
`'a'+'__'+classID+'__'+elementID+'__'+attributeID`
In this context there won't be an 'elementID' available so any integer can be used. Perhaps 'o' would be a good default value.

For checkboxes:
`'ck'+'__'+'a'+'__'+classID+'__'+elementID+'__'+attributeID`
In this context there won't be an elementID available so any integer can be used.

For operators:
`'aoper'+'__'+attributeID.`

As a follow on to this project we will serialize the user's selections into a different JavaScript object. We will also integrate this with the HTML user interface mockup that was created in the first phase.

With respect to the mockup UI, when the user selects (or drags from the tree and drops on the right panel) a class, as a result of the selection/drop, we will create the above form as one of a potential "stack" of such forms in the right hand panel. This may require a server round trip to retrieve the meta data object. More details on the exact work flow to follow.