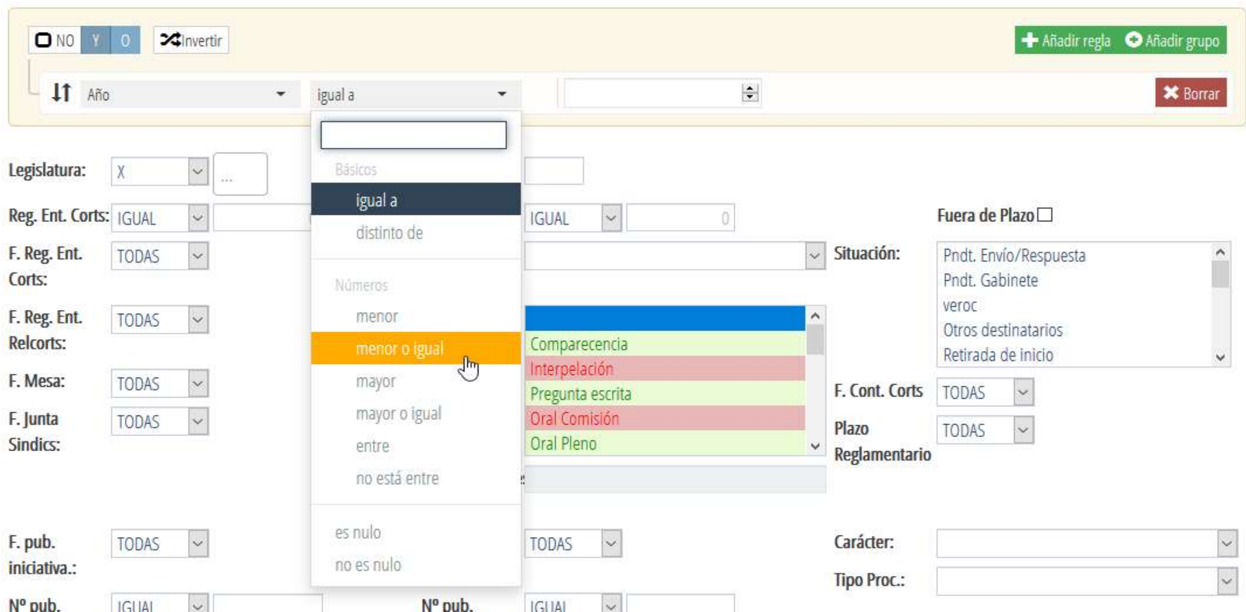
CASO DE ESTUDIO: BÚSQUEDA AVANZADA DE LA RAMA 5.1.X DE GVHIDRA

En las aplicaciones GASPAR y GV-MUSEIA surgió la necesidad de tener una búsqueda más compleja que la básica que ofrecce gvHIDRA (correspondencia campo de la TPL con campo de la BD). La búsqueda avanzada debe permitir el uso de operadores para realizar búsquedas más refinadas, incluso el poder definir funciones particulares adaptadas a cada caso.

La búsqueda avanzada se basa en el plugin JavaScript QueryBuilde (https://querybuilder.js.org) por lo tanto gvHIDRA en la versión 5.1.x ya se encarga de tener cargadas las librerías necesarias.

La implementación de la búsqueda avanzada afecta a la clase manejadora, la TPL y el views, para definir a qué campos se les aplica la búsqueda avanzada y operadores, y definir el entorno del editor de filtros. Lógicamente este tipo de búsqueda solamente será implementada en un panel filtro.



1. IMPLEMENTACIÓN DEL FILTRO EN LA TPL

La búsqueda avanzada comprende el uso de dos plugins: ***cweditorfiltros*** y ***cwfiltro***.

El plugin ***cweditorfiltros*** es de tipo *block*, donde se define la configuración general del editor de filtros. Dentro del bloque ***cweditorfiltros*** se irán añadiendo todos los campos que se quieran incluir en el editor de filtros con el plugin ***cwfiltro.***

```
{cweditorfiltros id="fil_editorFiltro"
      class                  = "gvh-querybuilder"
      plugins                = $editorFiltro_plugins
      allow_empty            = "true"
      claseManejadora        = "claseM"
      value                  = $defaultData_claseM.fil_editorFiltro
}
      {cwfiltro nombre="fil_ANYO"
            value                  = $defaultData_claseM.fil_ANYO
            dataType               = $dataType_claseM.fil_ANYO
            default_operator       = "equal"
            operators              = $operators_claseM.fil_ANYO
            label                  = #smty_Anyo#
      }
      …
…

{/cweditorfiltros}
```

<u>PARÁMETROS DE LOS PLUGINS</u>:

- **cweditorfiltros**:
   - *<u>id</u>*: Identificador del editor de filtros
   - *<u>class</u>*: Selector css que se aplicará en el editor de filtros (*gvh_querybuilder*)

- *claseManejadora*: Clase manejadora en la que se encuentra el editor de filtros.
- *plugins*: Array de plugins para extender las posibilidades de QueryBuilder. Esta variable smarty se define y asigna en el views.
- *allow_empty*: true/false. Si está a true permite que el editor no de error cuando no se introduce ningún campo rellenado
- *value*: Valor por defecto que se le puede asignar al campo. Será una variable smarty en la tpl, definida de la siguiente forma:

  `value = $defaultData_ClaseManejadora.NombreCampo`

- **cwfiltro**
  - PARÁMETROS OBLIGATORIOS:
    - *nombre*: Identificador del campo
    - *value*: Valor por defecto que se le puede asignar al campo. Será una variable smarty en la tpl, definida de la siguiente forma:

      `value = $defaultData_ClaseManejadora.NombreCampo`

    - *dataType*: Matriz con una estructura definida en la clase del panel, definirá el tipo de dato a mostrar en el campo (cadena, número, fecha…) y sus propiedades (longitud, obligatorio…) Será una variable smarty en la tpl, definida de la siguiente forma:

      `dataType = $dataType_ClaseManejadora.NombreCampo`

  - PARÁMETROS VOLUNTARIOS:
    - Con estos parámetros se podrán definir algunas reglas del editor de filtros. El listado de palabras clave válidas para estos parámetros, podemos encontrarlas en `https://querybuilder.js.org/#usage` (por ejemplo: *default_operator, placeholder, label...*)

Ejemplo de cwfiltro:

```
{cwfiltro nombre="fil_NUM_LEG" input="select" plugin="selectize" multiple="true"
value=$defaultData_claseM.fil_NUM_LEG    dataType=$dataType_claseM.fil_NUM_LEG
operators=$operators_claseM.fil_NUM_LEG label=#smty_Label#}
```

En este caso definimos que el filtro para "*fil_NUM_LEG*" sea de tipo lista (parámetro ***input***), utilizando el plugin selectize (parámetro ***plugin***), y también le indicamos que la lista debe ser múltiple (parámetro ***múltiple***). El atributo ***operators*** será un array con el listado de tipos de filtro para ese campo que se define en la clase manejadora.

2. IMPLEMENTACIÓN DEL FILTRO EN LA CLASE MANEJADORA

Principalmente la clase manejadora debe extender de la clase gvHidraFormAdvanced_DB en vez de la estándar gvHidraForm.

```
class MiClaseManejadora extends gvHidraFormAdvanced_DB
```

Normalmente la búsqueda se basa en comparaciones directas entre campo de la TPL y campo de la BD con el matching:

```
$this→addMatching("campoTPL","campoBD","tabla");
```

En el caso de que se quiera incluir el editor de filtros, se debe añadir la definición de filtro avanzado para ese campo. Por lo tanto ya no se utilizará la función addMatching() y se utilizará el addAdvancedFilterField():

```
$this->addAdvancedFilterField ($campoTPL, $matching, $tablaBD, $queryMode, $idQueryBuilder);
```

- ***$campoTPL***: Nombre del campo en la tpl correspondiente
- ***$matching***: Este parámetro contendrá con qué debe hacer matching el campo de la tpl. Hay varias opciones:
  - Nombre del campo en la BD.
  - Una expresión que deba cumplir ese campo (p.ej. Sql) o función.
  - Se puede dejar en blanco.
- ***$tablaBD***: Nombre de la tabla a la que corresponde el $campoBD en el caso de que éste sea un campo de la BD.
- ***$queryMode***: Entero entre 0 y 2 que indica el modo de consulta deseado.
- ***$idQueryBuilder***: Corresponde con el nombre del componente QueryBuilder en la tpl

El filtro que añadimos sobre un campo trabajará por defecto con todos los operadores, que son los siguientes:

- equal // not_equal
- in // not_in
- less // less_or_equal
- greater // greater_or_equal
- between // not_between
- begins_with // not_begins_with
- contains // not_contains
- ends_with // not_ends_with
- is_empty // is_not_empty
- is_null // is_not_null

Si lo que se quiere es particularizar esta lista de operadores se debe utilizar el siguiente método:

```
$this->addAdvancedFilterOperators ('$campoTPL', $operators[]);
```

- ***$campoTPL***: Nombre del campo en la tpl correspondiente

- ***$operators***: Enumerado de los operadores que se quieren utilizar. Por ejemplo: [‘equal’,’not equal’]

Puede ser que se necesite que el operador de búsqueda corresponda a una función particular para ese campo, en este caso tenemos el siguiente método:

```
$this->addAdvancedFilterFunction ('$campoTPL', '$fnFiltro', $operators[]);
```

- ***$campoTPL***: Nombre del campo en la tpl correspondiente
- ***$fnFiltro***: nombre de la función que implementará el operador específico para ese campo.
- ***$operators***: Enumerado de los operadores que se quieren utilizar. Por ejemplo: [‘equal’,’not equal’]

3. IMPLEMENTACIÓN DEL FILTRO EN EL VIEWS

En el views se deben definir los plugins, correspondiente al parámetro ***plugins*** del ***cwfiltro***. Se crea un array tipo FBObject() en el que se irá configurando qué plugins se necesitan para extender las posibilidades de QueryBuilder que interesen. Más información en https://querybuilder.js.org/plugins.html

Por ejemplo:

```
$plugins = new FBObject();
$plugins['bt-tooltip-errors'] = [ 'delay' => 100 ];
$plugins['sortable'] = null;
$plugins['filter-description'] = [ 'mode' => 'inline' ];
$plugins['bt-selectpicker'] = [ 'liveSearch' => true ];
$plugins['unique-filter']  = null;
$plugins['bt-checkbox']  = [ 'color' => 'primary' ];
$plugins['invert']  = null;
$plugins['not-group']  = null;

$s->assign( 'editorFiltro_plugins', $plugins );

{cweditorfiltros id="fil_editorFiltro" class="gvh-querybuilder" plugins=$editorFiltro_plugins allow_empty="true" claseManejadora="Piezas" value=$defaultData_Piezas.fil_editorFiltro}
```

4. USO DEL OPERADOR “similar”

Si nos interesa hacer uso del operador ***similar*** debemos tener en cuenta que hay que definirlo en la propia clase manejadora donde se encuentre el editor de filtros. Para ello se incluirá la función `getAdvancedOperators ($operator=null)` en la clase manejadora. Con ella se añade el operador ***similar*** al conjunto de operadores básicos que vienen por defecto.

La definición de esta función debe ser estática, ya que es invocada desde el método “construirWhereFromRule”, y puede que el objeto ya haya sido creado internamente por el framework.

```
/**
* @param    string $operator
* @return   BasicOperator|BasicOperator[]
*/
public static function getAdvancedOperators ($operator=null)
{

      $operators = (array) OperatorStandard::getStandardOperators (true);

      // Custom operator: 'similar'
      $operators['similar'] = new OperatorCustom ('similar', array (
                  'type'       =>     'similar' ,
                  'optgroup'   =>     'strings' ,
                  'nb_inputs'  =>     2 ,
                  'multiple'   =>     false ,
                  'apply_to'   =>     [ 'string' ] ,

                  'sql'        =>     ['processFn' =>
                               /**
                               * @param    IgepConexion $objConexion
                               * @param    integer             $queryMode
                               * @param    string       $field
                               * @param    array        $params
                               * @return   string
                               */
                               function ($objConexion, $queryMode, $field,
$params) {
                                      $value  = $params[0];
                                      $umbral = $params[1];

                                      $myQueryMode = ($queryMode % 10) + 10;
                                      return $objConexion->formatCondition
($field, $value, $myQueryMode, true, false, $umbral);
                               }
                  ] ,
                  'sqlOperator'       =>     "{ op: 'SIMILAR(?)', sep: ' BY
' }" ,
                  'lang_label'        =>     '#smty_labelSimilar#' ,
                  'default_value'            =>     [null, 70] ,
                  'allowUndiacritic'  =>     true
      ));


      if (isset ($operator)) {
            // Devolvemos solo los operadores deseados.
            if (is_array($operator)) {
                  // Devolvemos un conjunto de operadores deseados.
                  $results = array ();
                  foreach ($operator as $singleOperator) {
                         $results[$singleOperator] =
$operators[$singleOperator];
                  }
            } else {
                  // Devolvemos un único operador deseado.
                  $results = $operators[$operator];
            }
```

```
        } else {
            // Devolvemos todos los operadores.
            $results = $operators;
        }

        return $results;
    } // getAdvancedOperators
```