Query By Example Demo

To follow this demo, you need to have basic knowledge of Delphi

The Basic setup

Start a new Delphi project and add a Datamodule (File New -> Datamodule)

On the datamodule drop a TFDConnection, set it up to point to a database of your choice. I have in the demo set it to point to dbdemos.gdb located in Embarcaderos samples folder. This is an Interbase database. If you don't have Interbase installed and running, you can choose another database.

Add also to the datamodule a TFDQuery, connect it to the connection, and fill in some SQL:
SELECT * FROM BIOLIFE (or whatever works for you in your database)

Test the query if you feel like it

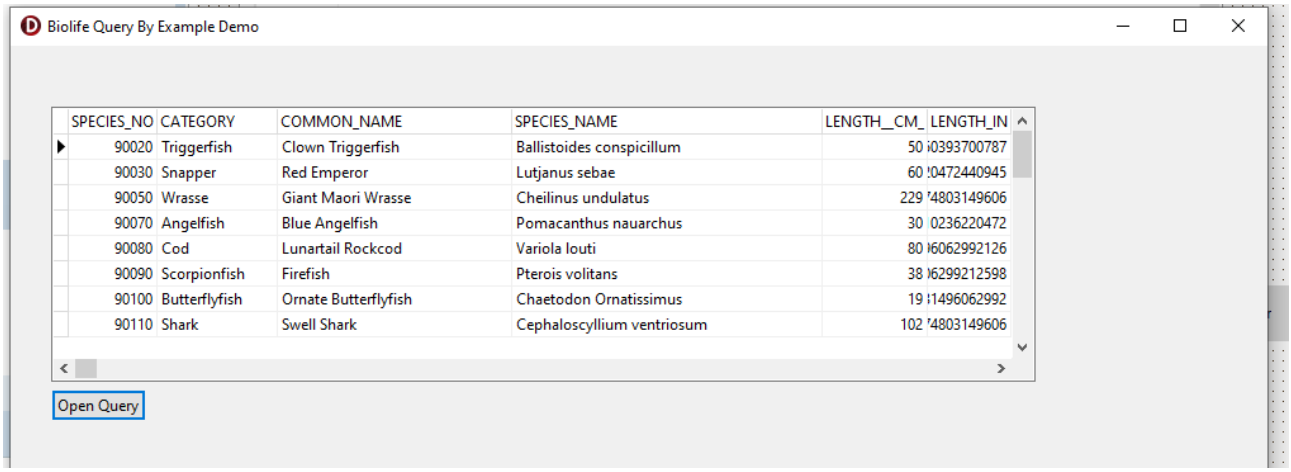
Add also here a TDatasource, connect to the query.

Go to the main form, and add your datamodule to the uses clause, and on the form add a TDbGrid, that you hook up to the datasource on the datamodule in the datasource property.

Add also a TButton, and add the code to open the query.

dmBiolife.qryBiolife.open;

If you have hooked everything up correctly your application should look similar to



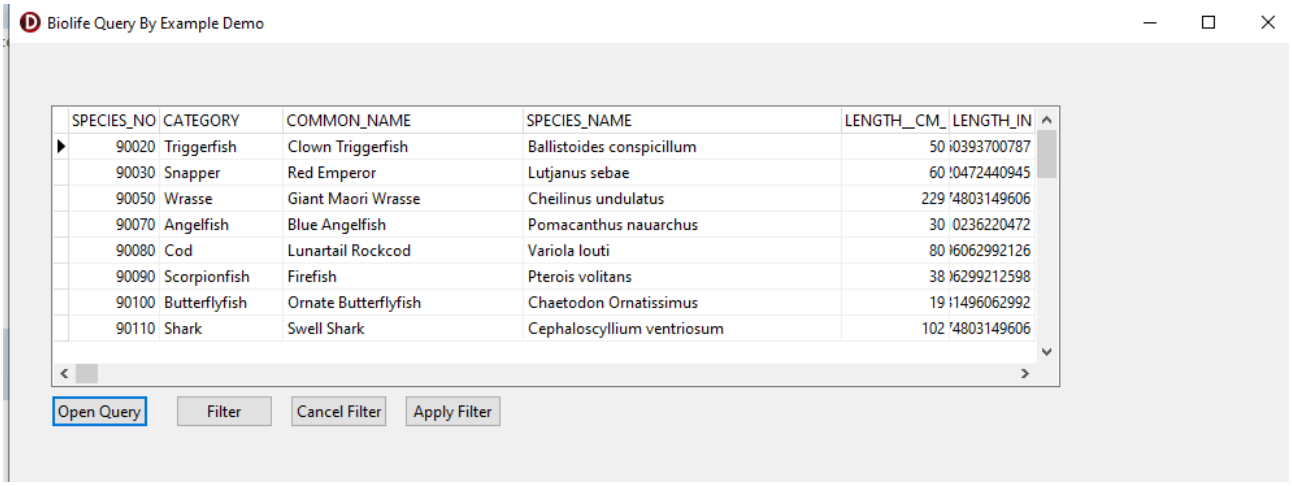
Adding Query By Example possibilities

The purpose of this demo is to show how a add Query By Example possibilities to your application.

This will give your user the ability to filter the above by writing SQL-like snippets. For example "> 80 " in the LENGTH_CM Column, or "%Angel% in COMMON_NAME.

To do this, add a TFDQBE component on the datamodule, and set the DataSource Property to point to the datasource that provides data to the grid.

On your main form, add three buttons



On the Filter Button add code:

dmBiolife.qbeBiolife.Edit; //This sets the filter in Edit-mode, so the User can write the filter

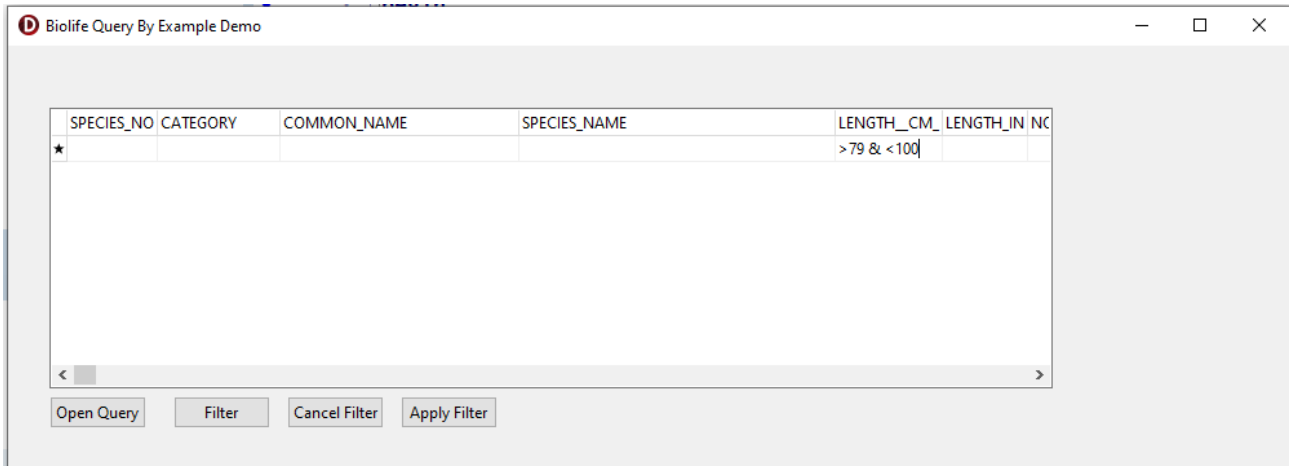
On the Cancel Filter button:

dmBiolife.qbeBiolife.Cancel; //which will cancel the filter

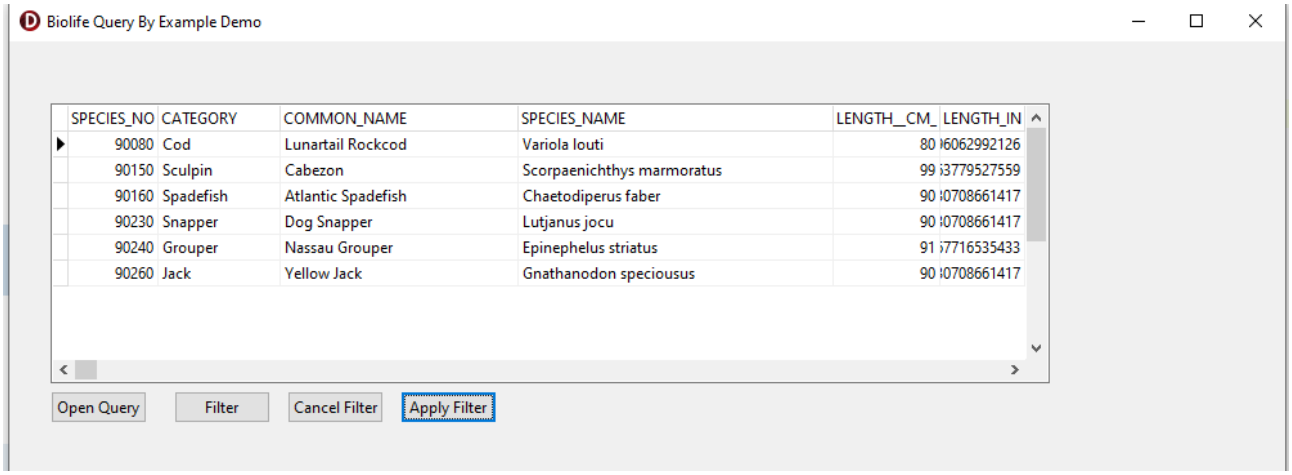
and finally, on the Apply Filter Button:

dmBiolife.qbeBiolife.Post; //Which will post the changes by the user and thus apply the filter

Hitting the "Filter" button will change the contents of the db grid. It now reflects the contents of the QBE component, rather than the Query component. The user can enter the filter directly in the grid



Will result in



When the Apply Filter button is clicked.

To cancel the filter, hit the cancel button.

Behind the scenes

The QBE component lets the user filter the query. What happens behind the scenes.

The QBE component has a Language property. By default it's set ti qlAuto, but you can also set it to qlSQL or qlFilter.

If it's set to qlAuto it will determine the language from the dataset type it is.

qlSQL is meant to work for TDFQueries, and qlFilter will work for all FireDAC datasets

qlSQL

As we have a query, it should by leaving the default setting use the qlSQL language.

Drop a Tmemo on the form

Create a method that will clear the memo and add the SQL in the query component

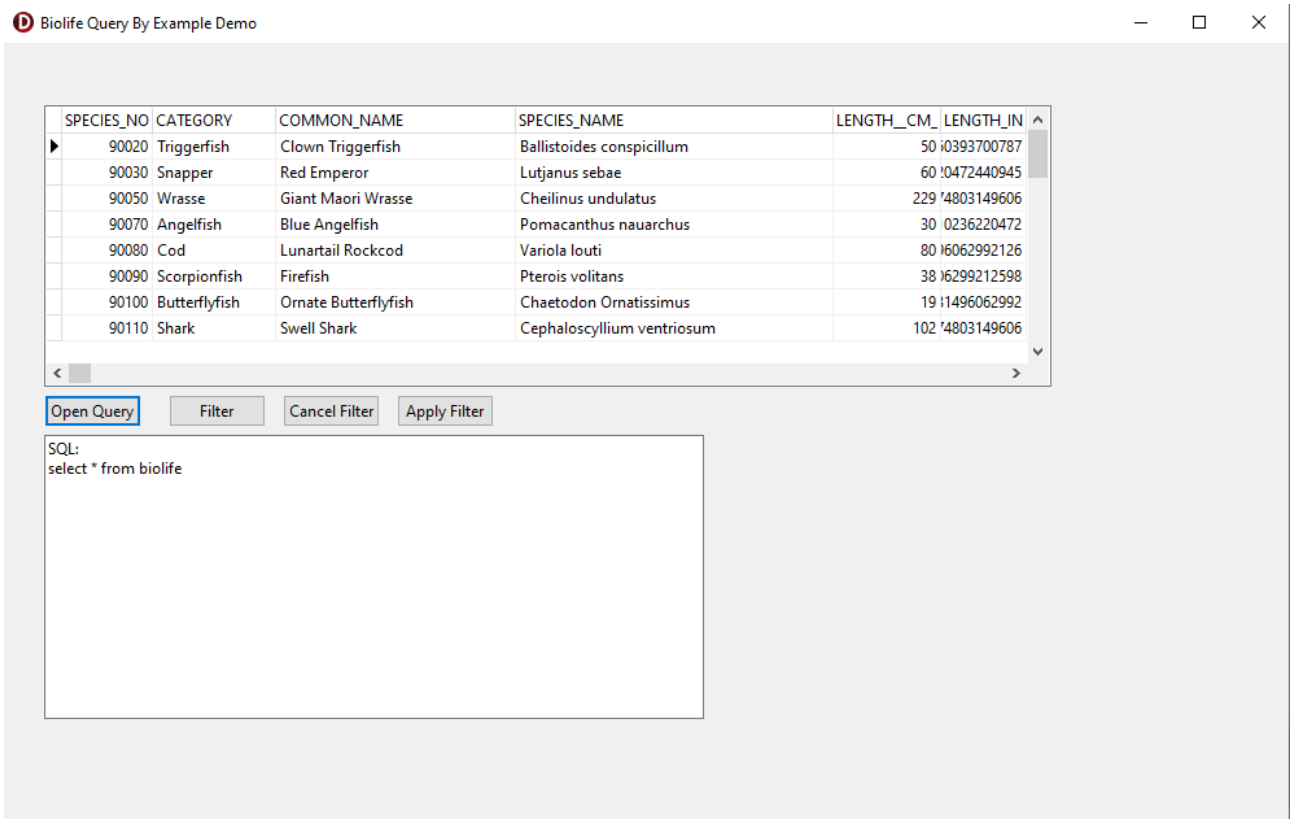
```
procedure TForm1.ShowSQL;

begin

 mmoSQL.lines.Clear;

 mmoSQL.Lines.Add('SQL:');

 for var i  := 0 to dmBiolife.qryBiolife.SQL.Count - 1 do

  mmoSQL.Lines.Add(dmBiolife.qryBiolife.SQL[i]);

end;
```

Call that method on all the buttons, like the Open Query Button

```
dmBiolife.qryBiolife.open;
ShowSQL;
```

Do the same for the Filter buttons

Here I have opened the query



And by setting the filter from before, I get the new SQL in the query:

SQL:

```
SELECT *
ROM (select * from biolife) A
WHERE ((((A.LENGTH__CM_ > 79 AND A.LENGTH__CM_ < 100)
))
)
```

qlFilter

Drop a TRadioGroup on the form, adding 3 options, set the ItemIndex to 0 and create a method to apply whatever language the user picks:

```
if dmBiolife.qbeBiolife.State <> qsInactive  then
 begin
   ShowMessage ('The filter must be inactive to change language');
   Exit;
 end;
```

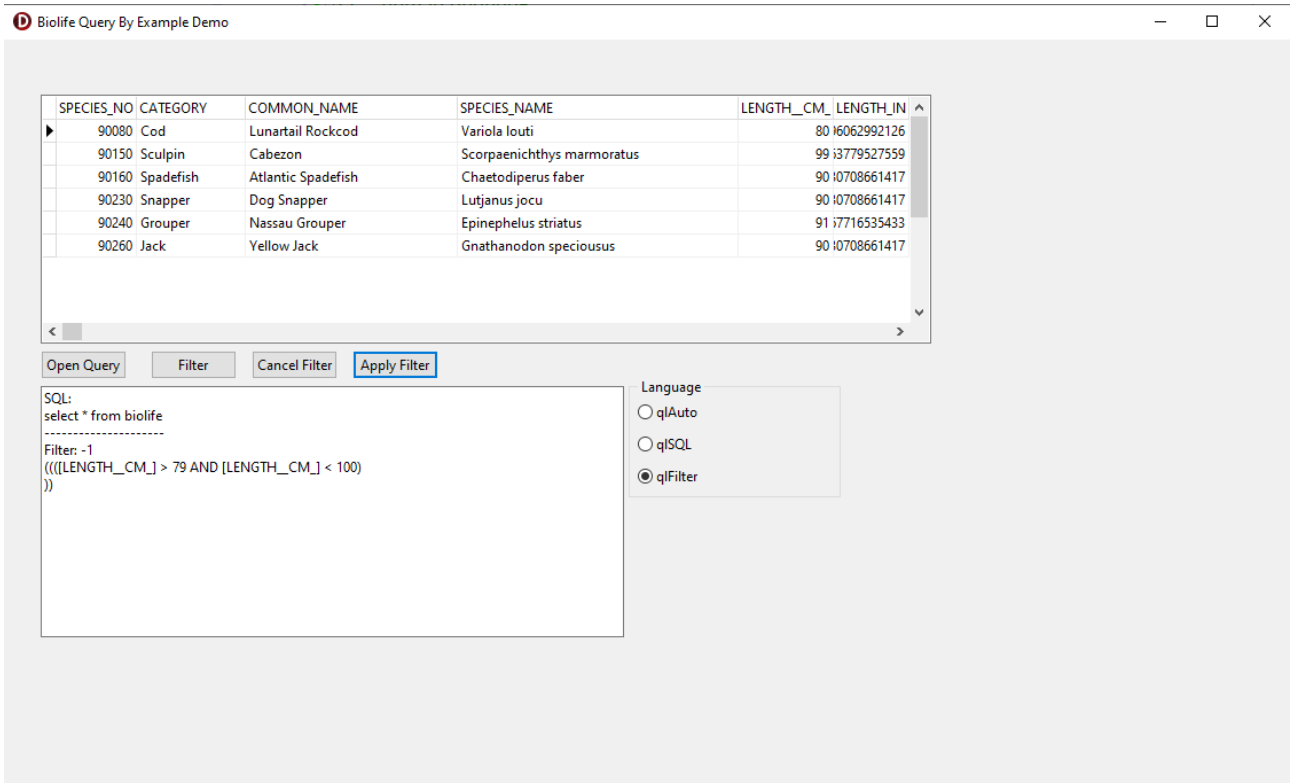
```
case rgLanguage.ItemIndex of
  0 : dmBiolife.qbeBiolife.Language := qlAuto;
  1 : dmBiolife.qbeBiolife.Language := qlSQL;
  2 : dmBiolife.qbeBiolife.Language := qlFilter;
  else
    ShowMessage('No language chosen');
end;
```

And call that on the radiogroups onClick event handler

Now add a couple of lines of code to the ShowSQL method:

```
mmoSQL.lines.Clear;
 mmoSQL.Lines.Add('SQL:');
 for var i  := 0 to dmBiolife.qryBiolife.SQL.Count - 1 do
   mmoSQL.Lines.Add(dmBiolife.qryBiolife.SQL[i]);
 mmoSQL.Lines.Add('----------------------');
 mmoSQL.Lines.Add(format('Filter: %s', [dmBiolife.qryBiolife.Filtered.ToString]));
 mmoSQL.Lines.Add(dmBiolife.qryBiolife.Filter);
```

By running the application now you can get the QBE component to use the good old “Filtered” property of the FireDAC dataset



Other options

The QBE component also offers various other options to explore.

Second TDBGrid

If for example you don't want the user to enter directly in the data grid, you can use TEdit (manually) TDBEdit or another TDBGrid.

On the datamodule, add a second datasource. No need to set any properties on it.

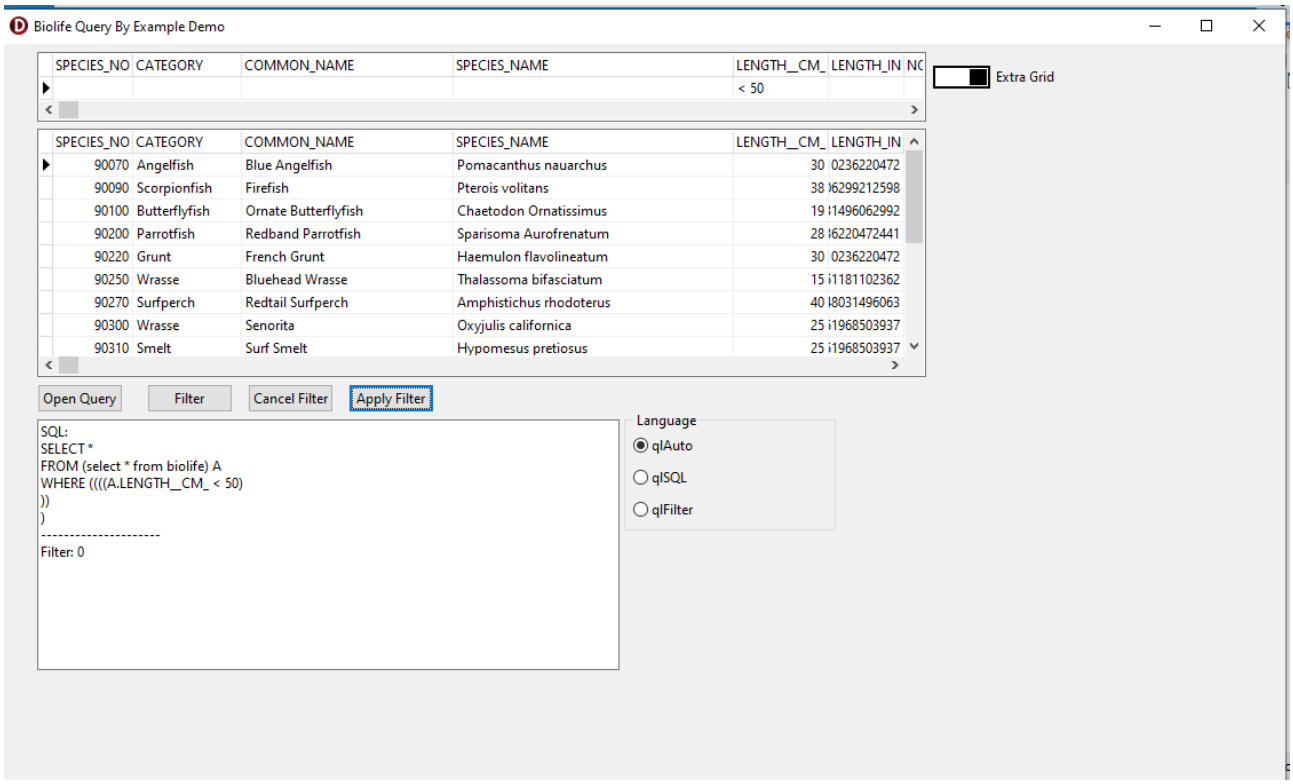
Add a second TDBGrid to the form, set the datasource to the datasource you just added.
Add also a TToggleSwitch with state captions set to:

CaptionOff := 'Main Grid';
CaptionOn := 'Extra Grid';

On the ToggleSwitch OnClick eventhandler set the code:

```
case tsGrids.State of
  tssOff: dmBiolife.qbeBiolife.QuerySource := nil ;
  tssOn: dmBiolife.qbeBiolife.QuerySource := dmBiolife. dsrcSecondGrid;
 end;
```

This will cause the SQE component to use the secondary grid for inputting the filter



Which allows the user to see the data while setting the filter, and also see which filter is applied

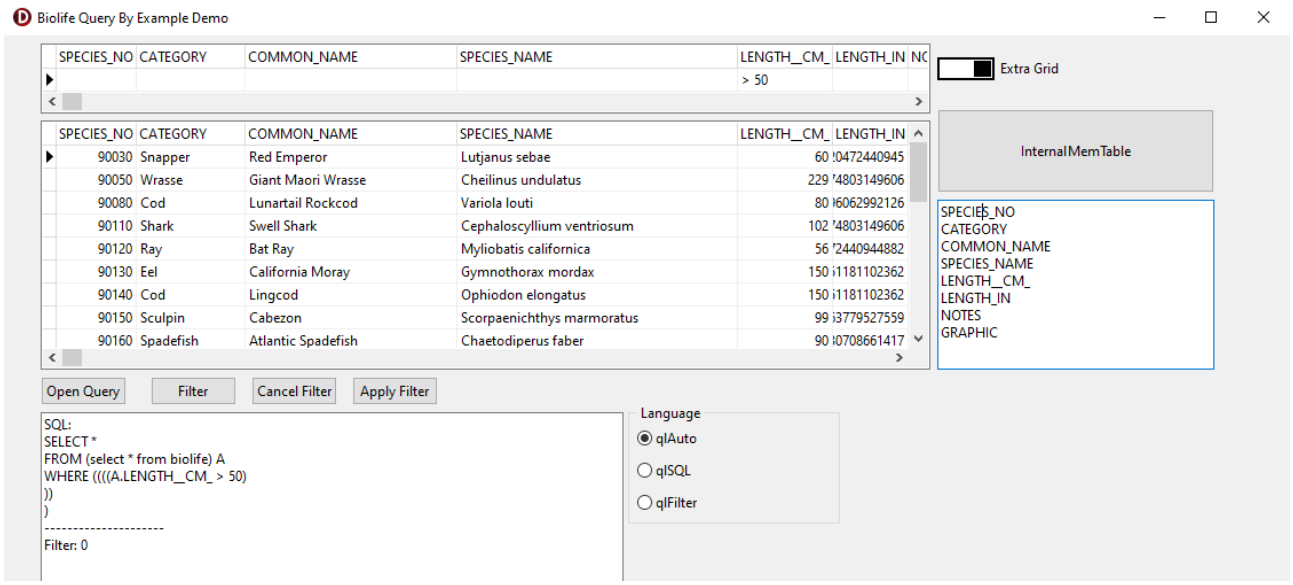
Technical Information

The filter that can be applied on the QBE component works on an internal FDMemTable on the component itself.

Just like a lot of the other FireDAC components consist of multiple other components, so does the TFDQBE

Just to see this, add a button and a Memo to the form, and add code to the buttons eventhandler:

```
if dmBiolife.qbeBiolife.State in [TFDQBEState.qsActive] then
 begin
  var sl := TStringList.Create;
  try
   for var i := 0 to dmBiolife.qbeBiolife.QBETable.FieldCount - 1 do
    begin
     sl.Add(dmBiolife.qbeBiolife.QBETable.Fields[i].FieldName);
    end;
  finally
   mmoInternalFields.Lines.Assign(sl);
   sl.Free;
  end;
 end;
```



And we can see that the fieldnames of the internal memtable actually match the fieldnames of the table we are working on.

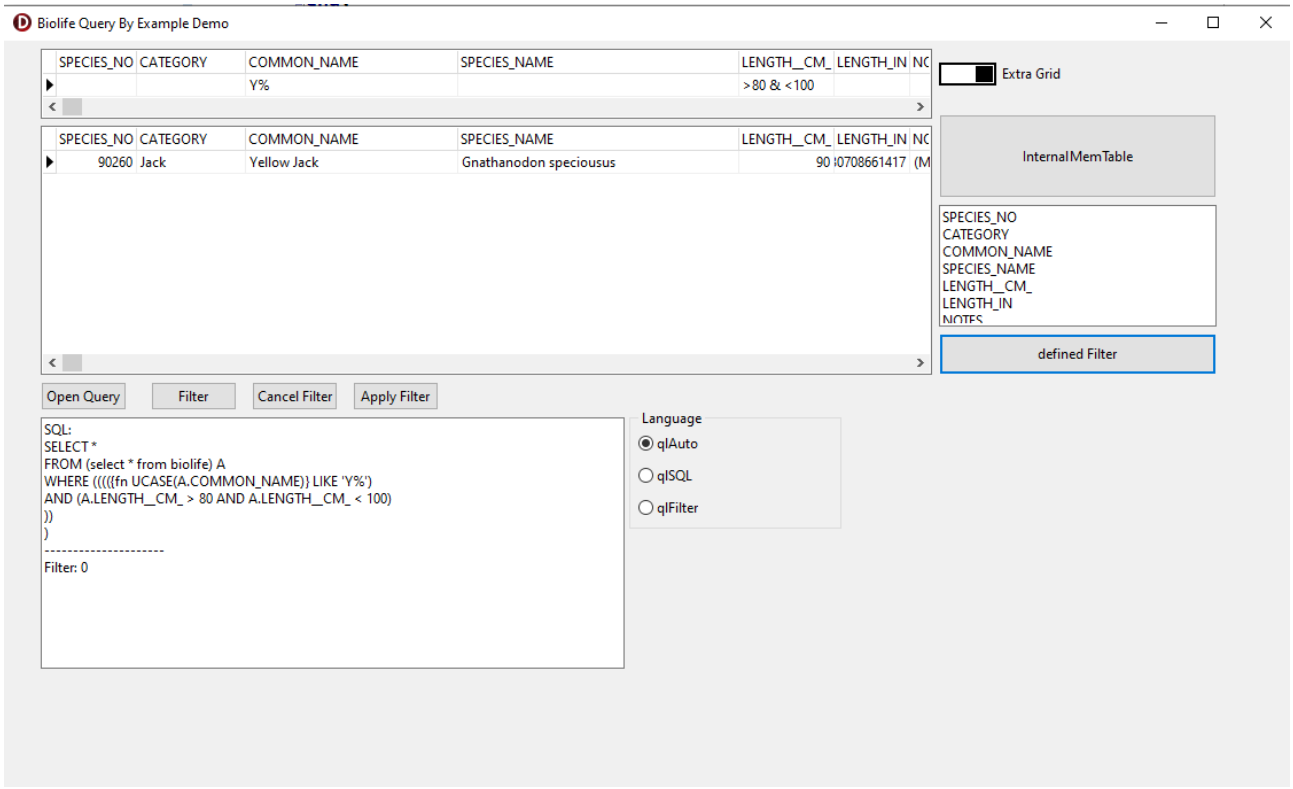
Predefined filters

Now that we know that there is an internal memtable, we can apply that knowledge and have predefined filters

Add a button to the form, and add code:

```
dmBiolife.qbeBiolife.Edit;
dmBiolife.qbeBiolife.QBETable.EmptyDataSet;
dmBiolife.qbeBiolife.QBETable.Append;
dmBiolife.qbeBiolife.QBETable.FieldByName('LENGTH__CM_').AsString := '>80 & <100 ';

dmBiolife.qbeBiolife.QBETable.FieldByName('COMMON_NAME').AsString := 'Y%';
dmBiolife.qbeBiolife.QBETable.Post;
dmBiolife.qbeBiolife.Post;
ShowSQL;
```

Which will show us all fish that are larger than 80 CM, smaller than 100 CM and starts with “Y”



Save To file – Load from file

The filters that are set can also be saved to or loaded from a file.

Drop a couple of buttons to the form, with code like:

```
dmBiolife.qbeBiolife.SaveToFile('c:\temp\qbe.txt');

dmBiolife.qbeBiolife.LoadFromFile('c:\temp\qbe.txt');
```

And the saved text file will represent the filter.

```
[{"COMMON_NAME":"Clown","LENGTH__CM_":"<80"
```