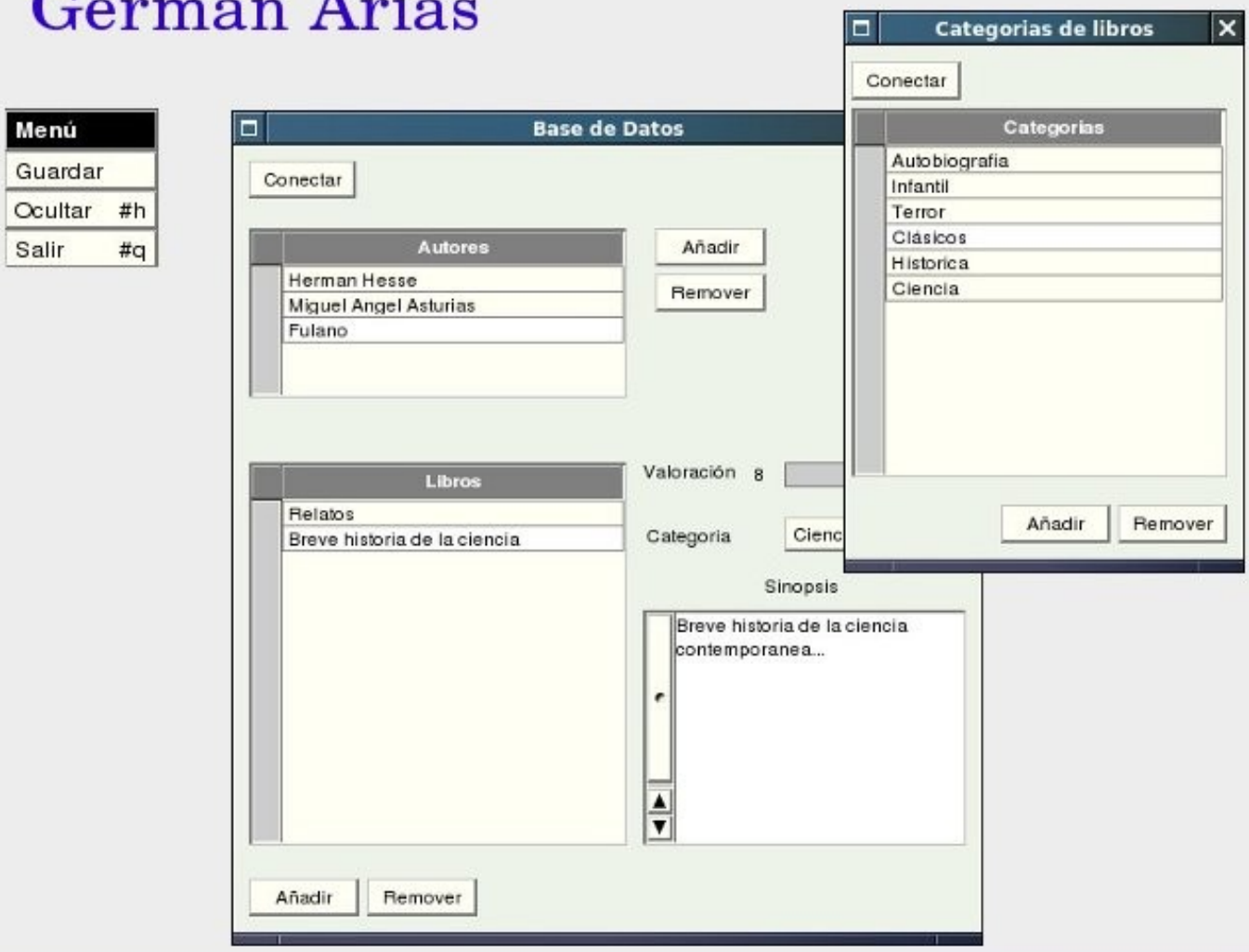
GDL2

(GNUstep Database Library 2)

Germán Arias



Una guía paso a paso para crear una sencilla aplicación cliente-servidor

GNUstep Database Library 2 (GDL2)

GDL2 es el framework de GNUstep para crear aplicaciones que utilizan bases de datos. Este framework contiene tres librerías:

- EOControl.
- EOAccess.
- EOInterface.

Y una aplicación llamada DBModeler.app para crear los modelos de las bases de datos. Modelos que posteriormente pueden ser usados en GORM para crear la interfaz gráfica de nuestra aplicación.

En los sencillos ejemplos que realizaremos aquí, no nos involucraremos con el manejo directo de estas clases, ya que DBModeler.app hará todo el trabajo por nosotros. Aunque si lo que se desea es utilizar GDL2 en una aplicación web, si es necesario el manejo directo de las clases, ya que no existe una aplicación gráfica para el desarrollo de dichas aplicaciones.

Instalación

Aparte de GNUstep y los gestores de bases de datos que deseemos utilizar, se requieren las librerías de desarrollo de estas bases de datos. Así como la librería **Renaissance** desde SVN, la cual puede obtenerse con:

svn co http://svn.gna.org/svn/gnustep/libs/renaissance/trunk/ Renaissance

Actualmente GDL2 solamente soporta bases de datos PostgreSQL y SQLite. Una vez instalados estos requisitos, en una terminal vamos a la carpeta de GDL2 para escribir:

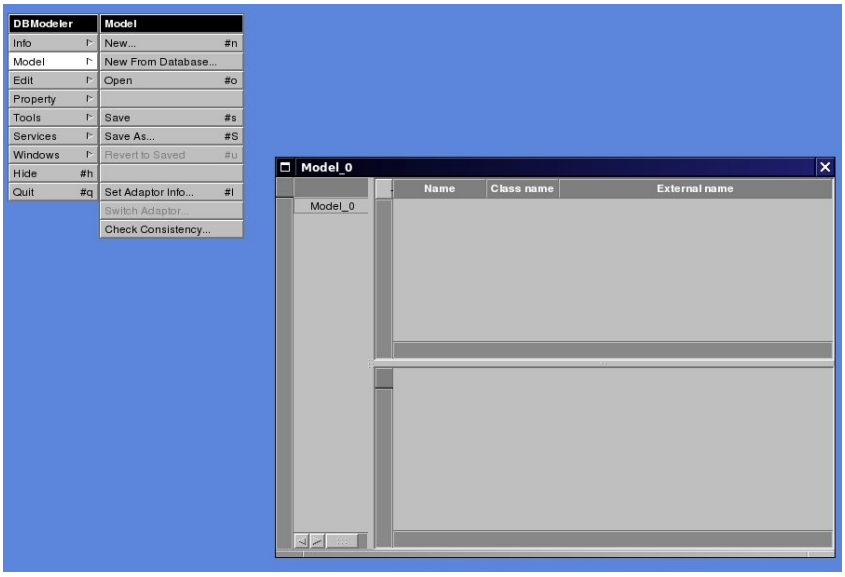
./configure

Terminada la configuración, hay que verificar que se hayan detectado los gestores de bases de datos instalados (Postgre o SQLite). Si todo esta bien damos un *make* y por último un *make install* (con permisos de superusuario) para instalar todo.

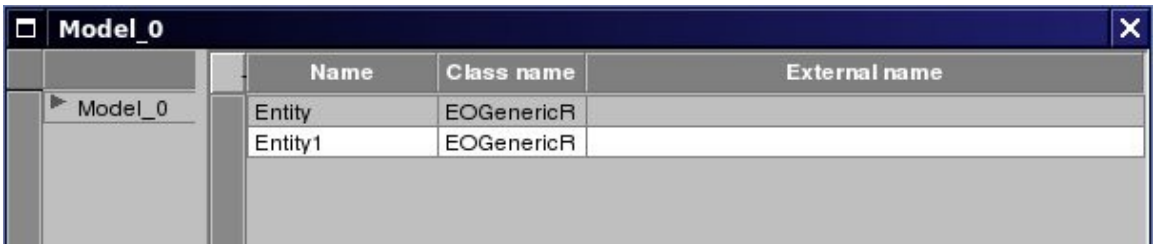
Lo único que nos falta ahora es cargar la paleta de GDL2 en GORM para que este pueda utilizar modelos creados con DBModeler. Para ello iniciamos GORM, y en el menú vamos a *Tools -> Load Palette*. Y en el panel que se abre, vamos a la ruta */usr/GNUstep/Local/Library/ApplicationSupport/Palettes/*, seleccionamos la paleta GDL2 y damos un clic en **Aceptar** para cargar la paleta. Esta nueva paleta no contiene ningún elemento, pero le permite a GORM utilizar GDL2.

Creando la Base de Datos

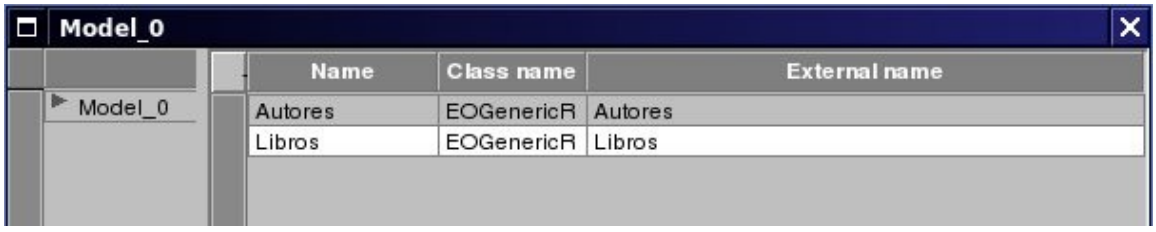
Antes de iniciar, vamos a crear una carpeta llamada *Biblioteca*, para guardar nuestro proyecto. Y dentro de esta, otra capeta llamada *Resources*. Ahora arrancamos DBModeler (es preferible hacerlo desde una terminal, para ver posibles errores) y en el menú seleccionamos *Model -> New...*. Esto abrirá el editor de tablas:



En el menú seleccionamos *Property -> Add Entity* dos veces, para agregar dos entidades (tablas) a nuestra base de datos.

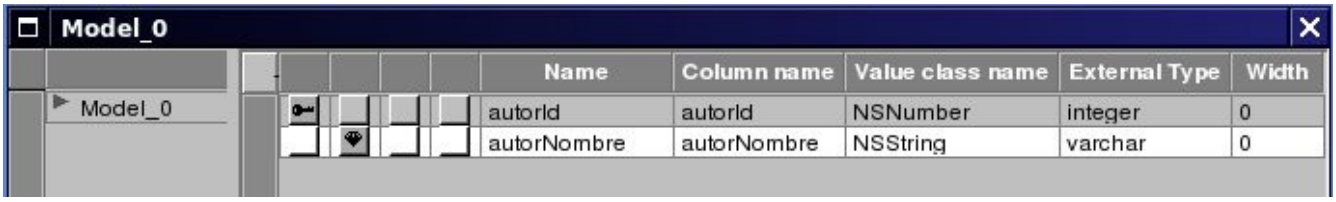


Editamos los nombres de estas tablas como se muestra en la siguiente imagen:



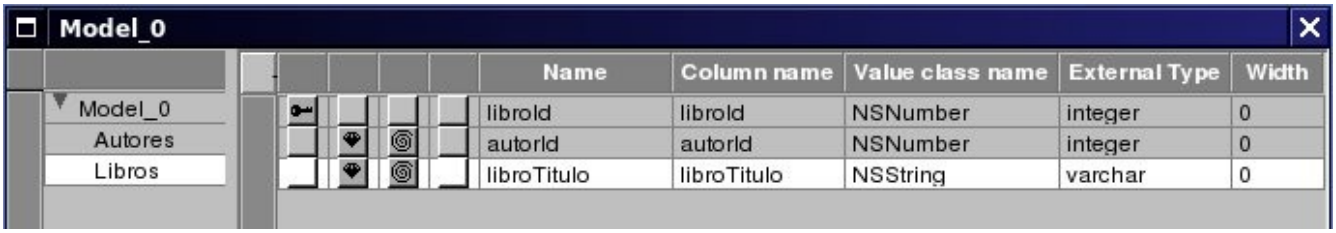
El nombre de la clase no lo modificamos, DBModeler se encarga de colocar el nombre de la clase que corresponde a una entidad (tabla). Vamos ahora a agregar los atributos (datos o columnas de la tabla) que contendrá cada una de las tablas. Seleccionamos primero la tabla Autores, y en el menú

seleccionamos *Property -> Add Attribute* dos veces, para agregar dos atributos a la tabla. Los cuales editamos como se muestra en la siguiente imagen:

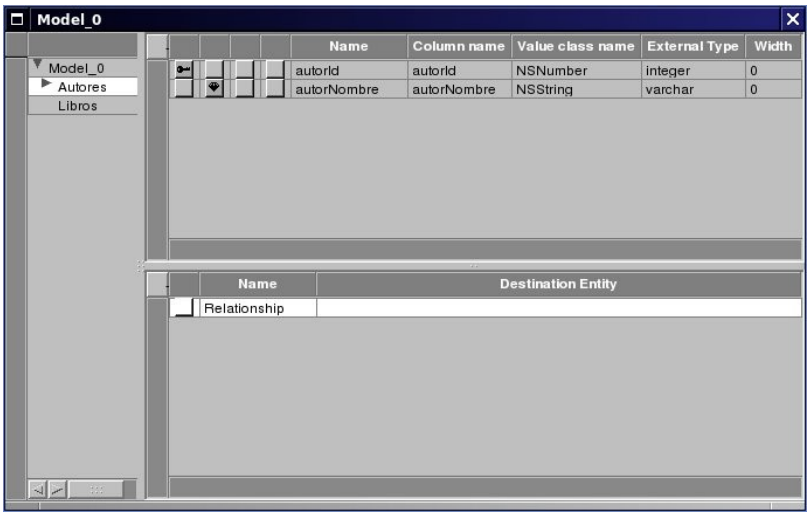


En la primera columna se establecen los datos que serán llaves (icono de llave), la segunda columna establece si se almacenara un objeto en la columna de la tabla (icono de ruby), la tercera establece si dicha columna permite valores nulos (icono de espiral), y la cuarta si la columna esta bloqueada (icono de candado). El objeto del resto de columnas es claro. Es importante hacer observar que si un dato será llave, entonces este dato no puede almacenar un objeto en la columna (es decir, que si tiene el icono de llave, entonces no puede tener el de ruby) aun cuando para este ejemplo, a la llave le asignamos la clase NSNumber. Otro aspecto a observar, es que el ancho de cada uno de los atributos esta establecido a cero en el editor de tablas, esto le indica a DBModeler que dicho atributo no tiene un límite de longitud.

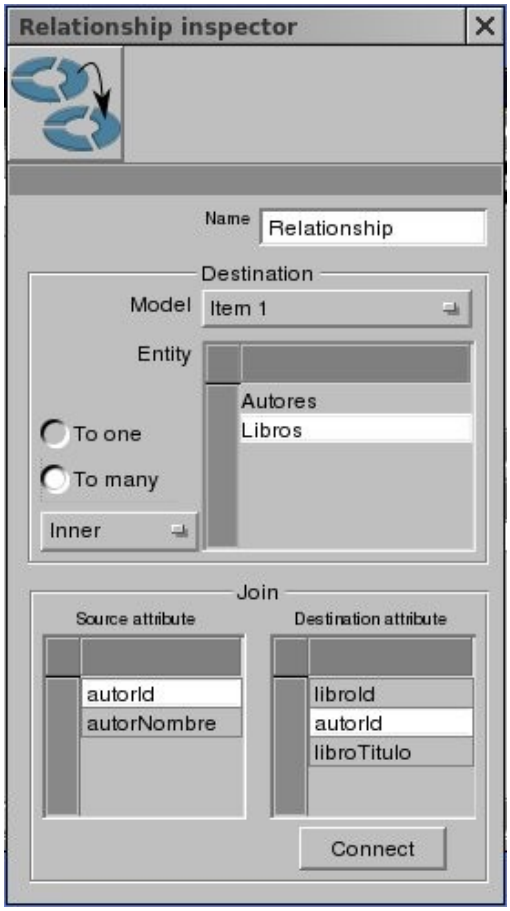
Los atributos para la tabla Libros, deben quedar de la siguiente forma:



Seleccionando ahora la tabla Autores, vamos a agregarle una relación *uno a varios* hacia la tabla Libros, ya que cada autor puede haber escrito varios libros. Para ello, teniendo seleccionada la tabla Autores, seleccionamos en el menú *Property -> Add Relationship*, lo que agregara una relación en la parte inferior del editor de tablas:

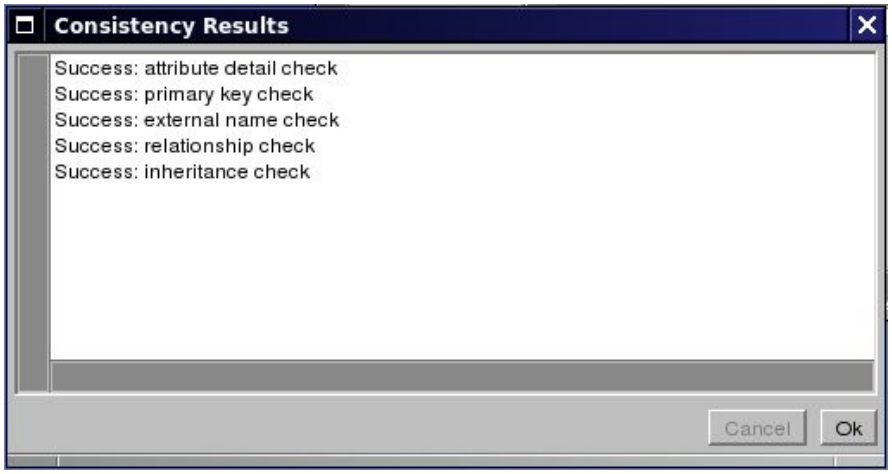


Seleccionando dicha relación, elegimos en el menú *Tools -> Inspector*, lo que abrirá el inspector de relaciones:

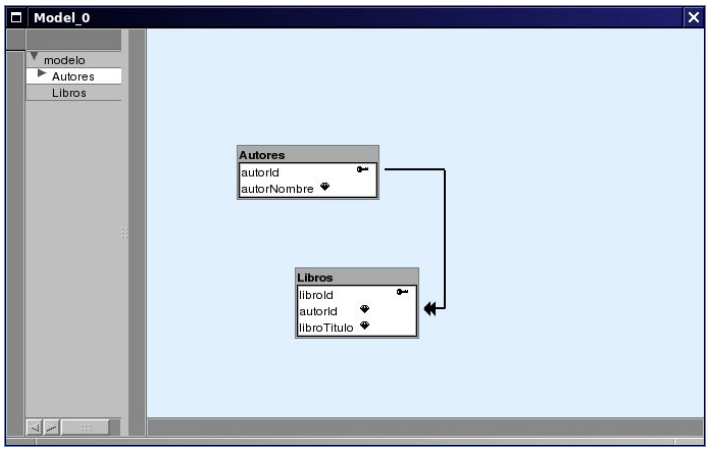


En la sección **Destination** seleccionamos la tabla hacia la cual queremos establecer la relación, es decir, la tabla Libros. Seleccionamos también la opción *To many*, ya que se trata de una relación *uno a varios*. Ahora en la sección **Join**, seleccionamos los atributos a relacionar, es este caso, el dato *autorId*. Damos entonces un clic en el botón *Connect* para establecer la relación. Cerramos ahora el inspector, y en el editor de tablas editamos la relación como se muestra en la imagen:

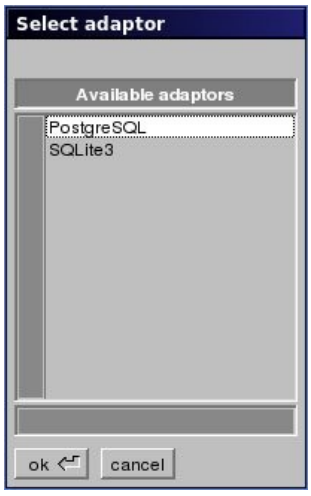
El nombre de la relación pudo establecerse también en el inspector. Verificamos ahora la consistencia de nuestra base de datos seleccionando en el menú *Model -> Check Consistency...*



Antes de proceder a ejecuta los correspondientes comandos SQL, podemos dar un vistazo a nuestra base de datos seleccionando *Tools -> Diagram Editor* (esta opción aun no funciona muy bien)

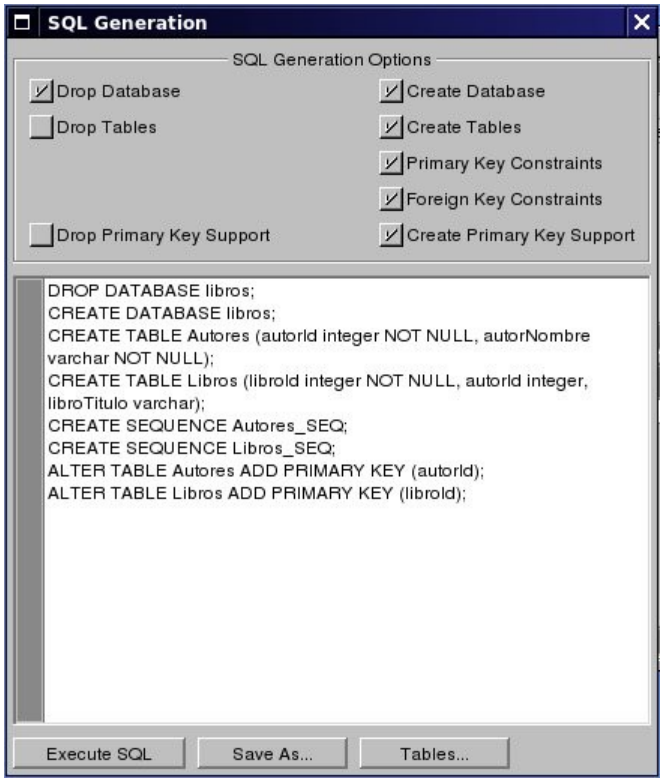


Para retornar al editor de tablas seleccionamos *Tools -> Table Editor*. Procedamos ahora a conectar con la base de datos en la cual trabajaremos. En el menú seleccionamos *Model -> Set Adaptor Info....* Y en el panel que aparece seleccionamos el gestor de bases de datos:



Y en panel que se abre introducimos la información correspondiente. Nuestro nombre de usuario, nuestra clave (si la hay), y la base de datos con la cual trabajaremos.

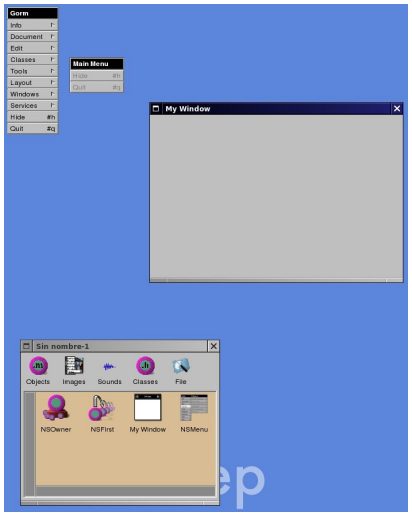
Para ejecutar comandos SQL seleccionamos en el menú *Tools -> Generate SQL*. En el panel que aparece seleccionamos las casillas, en el orden correspondiente, de lo que queremos que se ejecute. Para este ejemplo yo seleccione primero borrar la base de datos (porque ya había trabajado con ella anteriormente), y luego seleccione las opciones de la derecha de arriba hacia abajo. El orden en que se seleccione es importante, ya que es el orden en que se ejecutaran los comandos. Una vez seleccionado lo deseado, damos un clic en el botón *Execute SQL* para crear la base de datos. Verificamos en la terminal que todo se haya ejecutado correctamente. De ser así, cerramos el panel **SQL Generation** (para eliminar un comando SQL, simplemente se deselecciona la casilla correspondiente).



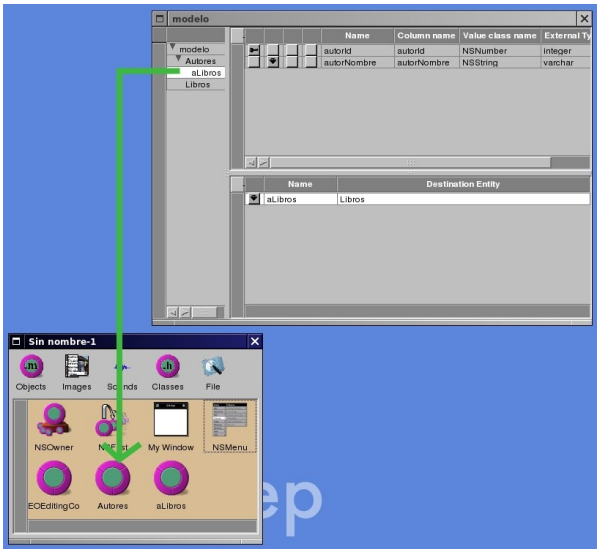
Solo nos queda guardar el modelo con el nombre **modelo** en la carpeta *Resources* de nuestro proyecto. Minimizamos ahora DBModeler para pasar a crear la interfaz gráfica de nuestra aplicación (DBModeler todavía nos sera útil, por esta razón no lo cerramos).

La interfaz de nuestra aplicación

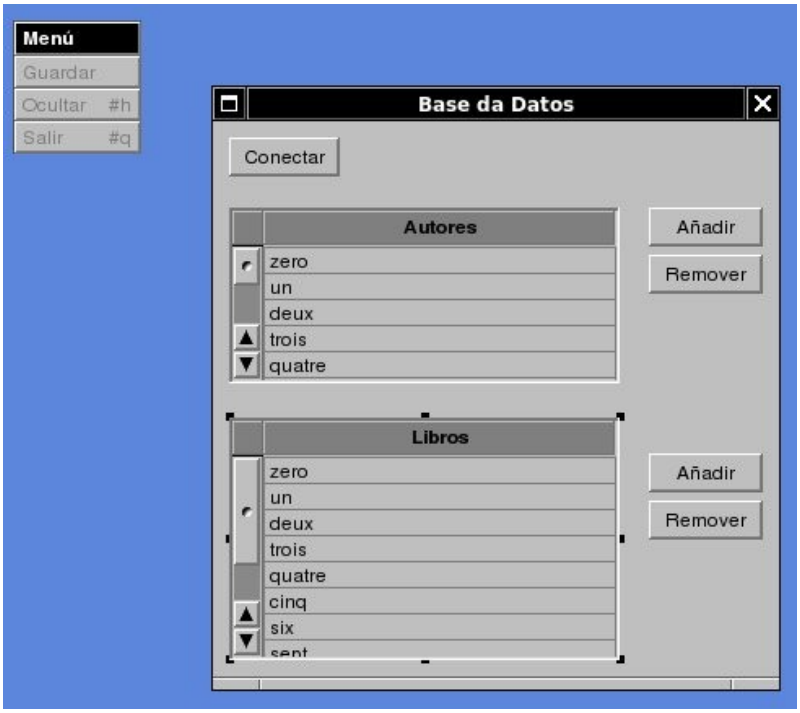
Abrimos Gorm y creamos una aplicación seleccionando *Document* -> *New Application* para iniciar nuestra interfaz.



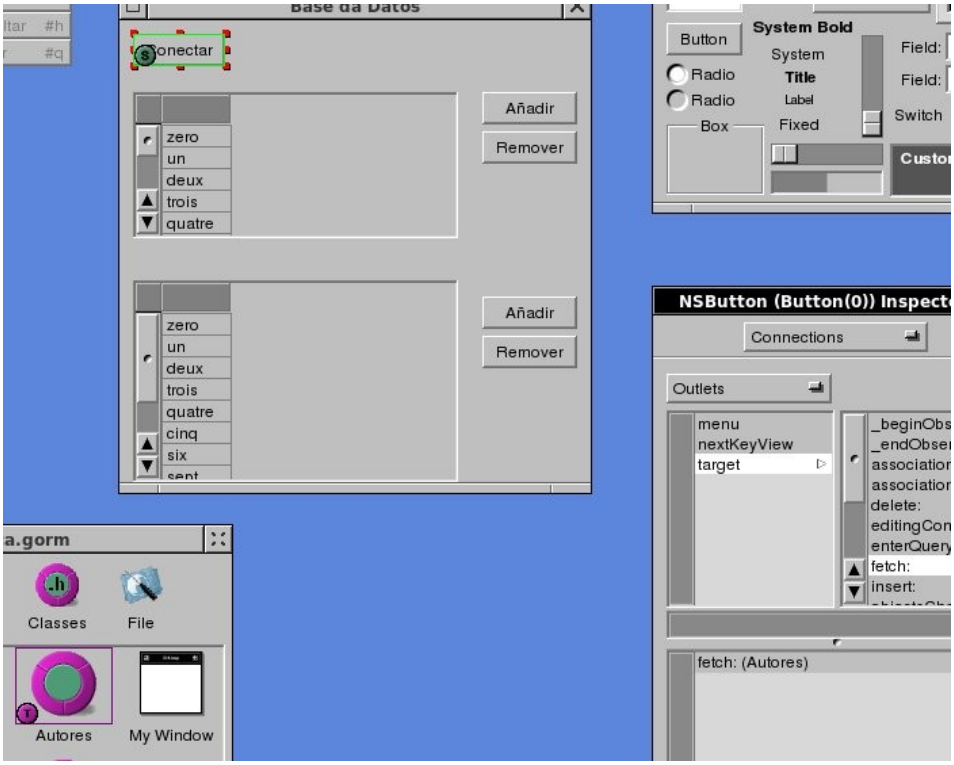
Vamos a agregar ahora el modelo hecho con GBModeler a nuestra aplicación. Para ello, abramos DBModeler y, seleccionando previamente la relación aLibros de nuestro modelo, la arrastramos con un clic izquierdo a nuestro documento gorm. Esto agregara tres objetos: **EOEditingContext**, **Autores** (la tabla autores) y **aLibros** (la tabla libros).



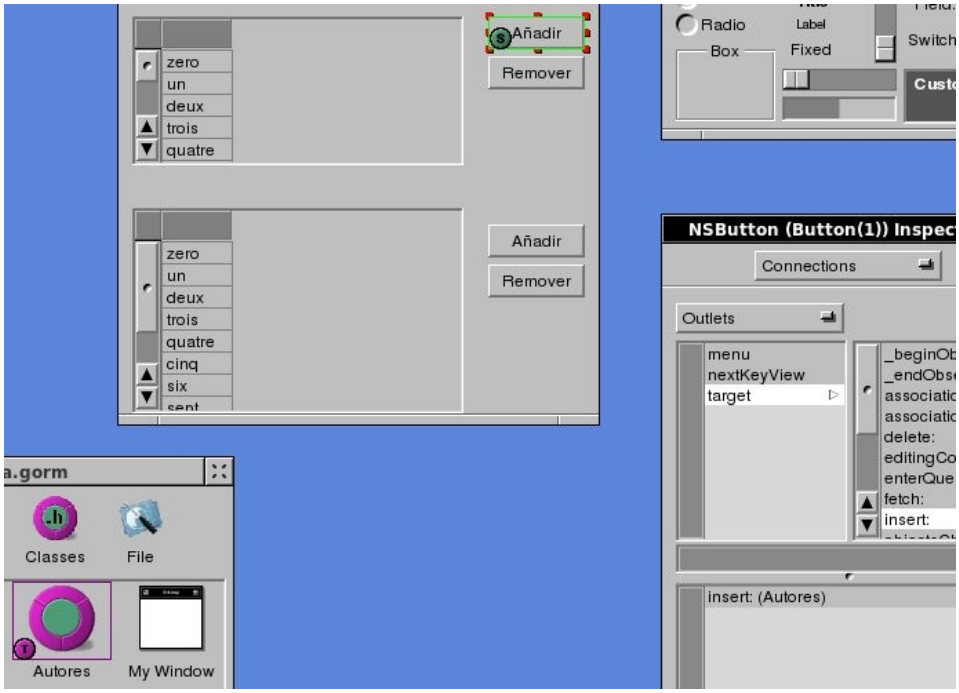
Ahora agregamos elementos a nuestra interfaz del forma que quede como se muestra en la imagen de abajo. Observese que las tablas solamente tienen una columna, con el ancho apropiado para ingresar datos, y que se agrego un nuevo ítem al menú, llamado **Guardar**.



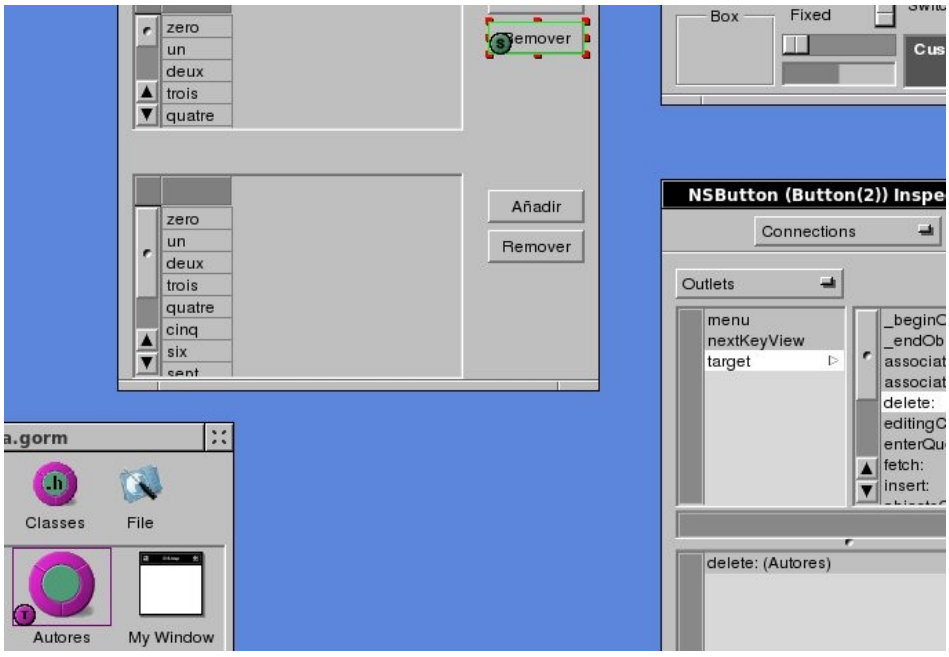
Comencemos conectando el botón **Conectar** con el objeto **Autores** y el método *fetch:*



Ahora conectemos el botón **Añadir** (el de arriba) también con el objeto **Autores** y con el método *insert:*

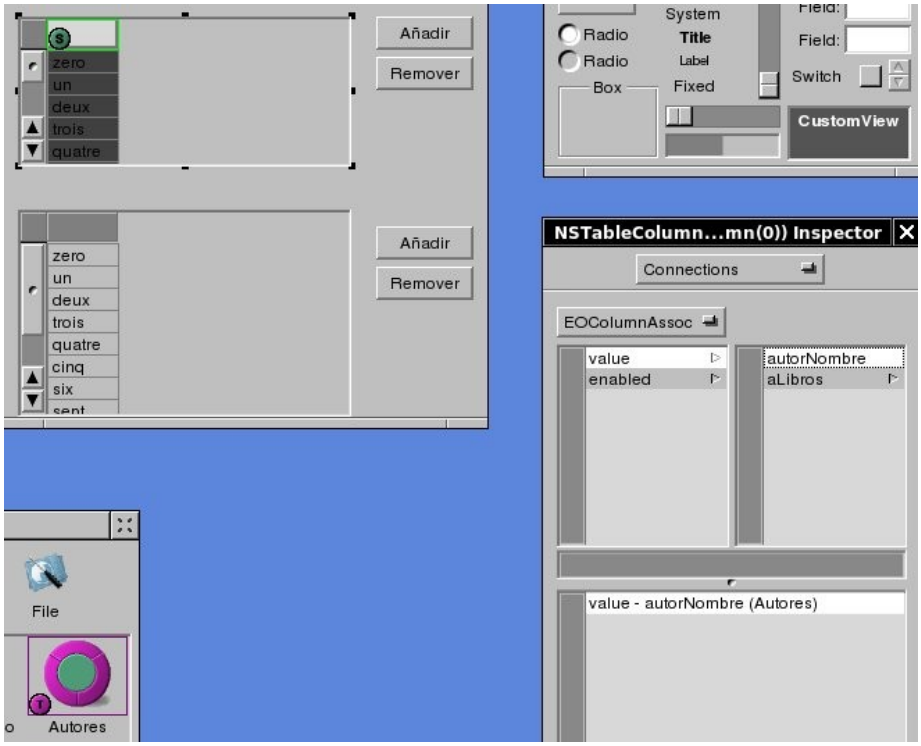


Y el botón **Remover** con el método *delete:* del mismo objeto:

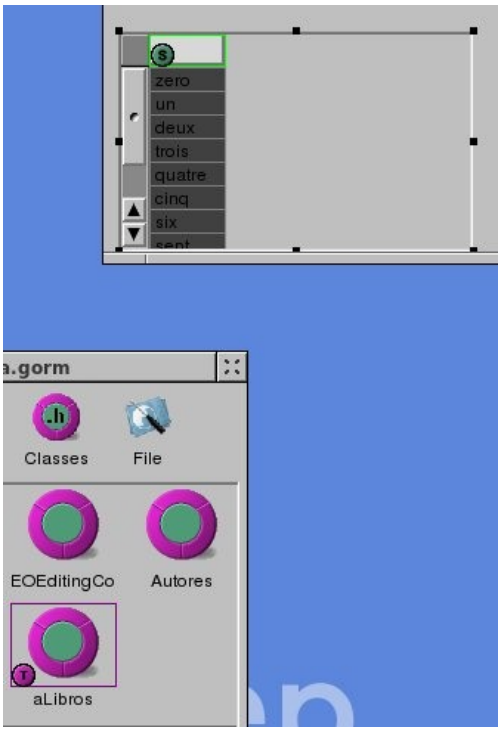


Para los botones de la tabla inferior (la de los libros) es lo mismo, sólo que las conexiones son con el objeto **aLibros**.

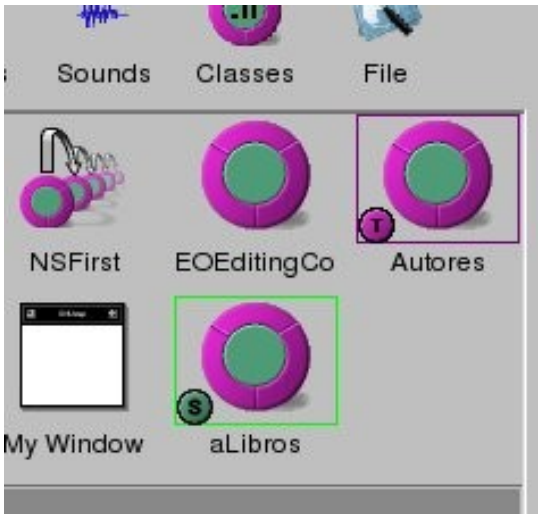
Ahora debemos conectar las tablas. En la de arriba, seleccionamos la columna dando un clic sobre la cabecera de dicha columna. Y luego realizamos la conexión arrastrando la columna (no la cabecera) al objeto **Autores**. Y en el Inspector seleccionamos **EOColumnAssoc** en lugar de **Outlets**, y seleccionamos dentro de *Value* el dato *autorNombre*.



De igual manera conectamos la columna de la tabla inferior. Solamente que la conexión se hace con el objeto **aLibros** y el dato *libroTitulo*

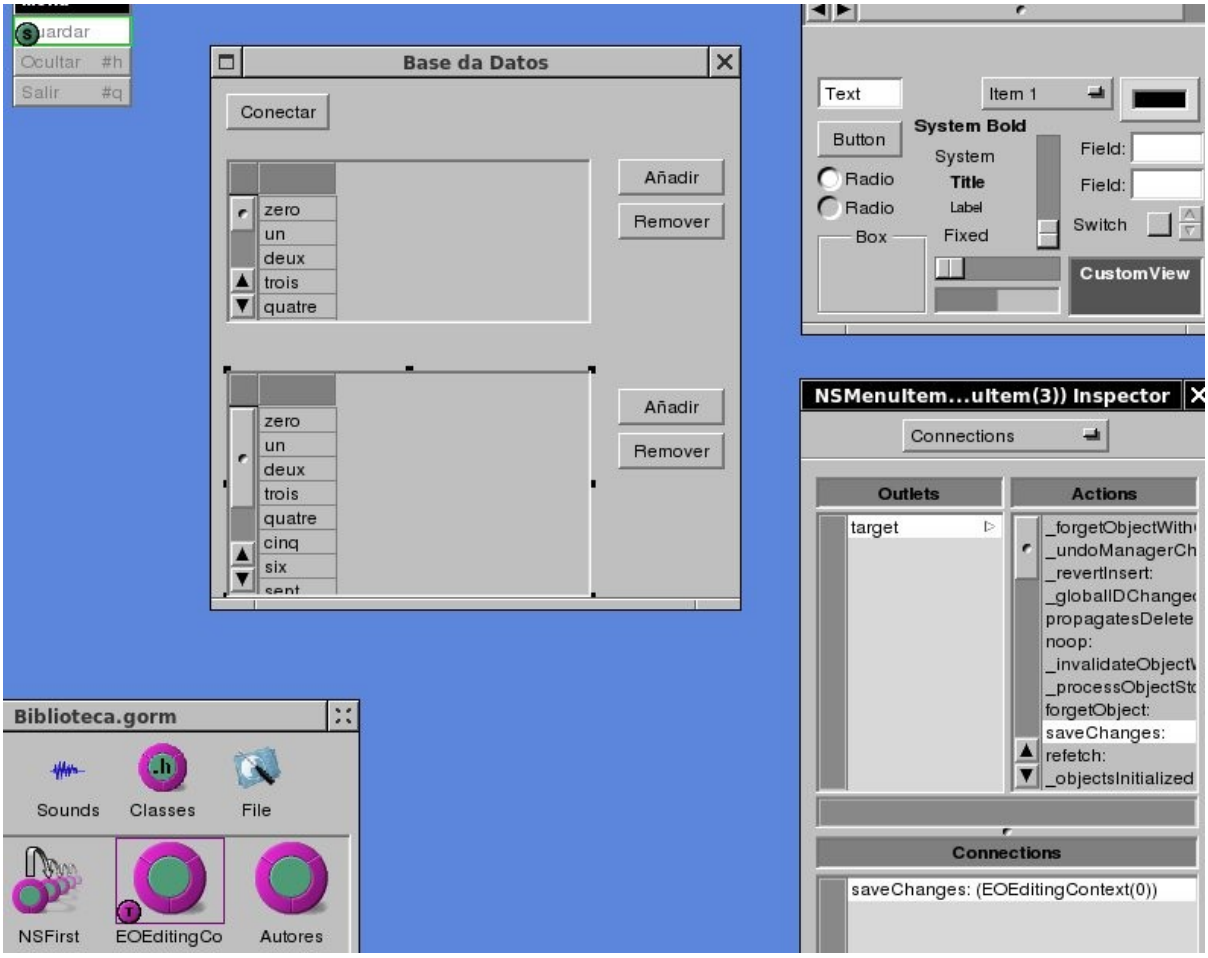


Ahora vamos a conectar la tabla inferior con la superior. De tal forma que la tabla inferior muestre los datos (los libros) correspondientes al dato seleccionado en la tabla superior (el autor). Para ello, conectamos el objeto **aLibros** con el objeto **Autores**.



Y en el Inspector seleccionamos, en lugar de Outlets, **EOMasterDetailAssoc**. Y dentro de *parent* seleccionamos *aLibros* para establecer la conexión:

Por último, vamos a conectar la opción *Guardar* del menú con el objeto **EOEditingContext** al método *saveChanges:*



Guardemos ahora nuestra interfaz con el nombre **Biblioteca** en nuestra capeta *Resources*.

Archivos necesarios

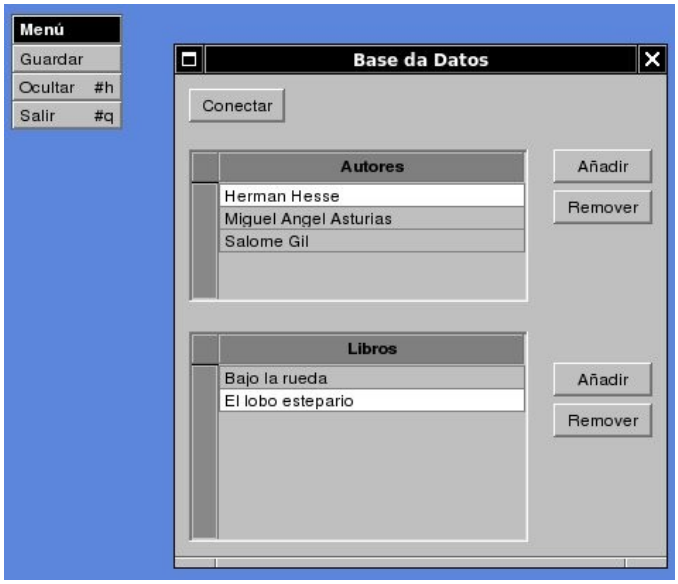
Necesitamos dos archivos para poder compilar nuestra aplicación. Ambos en nuestra carpeta *Biblioteca*. Al primero de ellos, lo llamaremos **main.m**, y contendrá la función *main* de nuestra aplicación:

```
#include <AppKit/NSApplication.h>

int main (int argc, const char *argv[])
{
  return NSApplicationMain(argc, argv);
}
```

Al segundo archivo lo llamaremos *GNUmakefile* y contendrá lo siguiente:

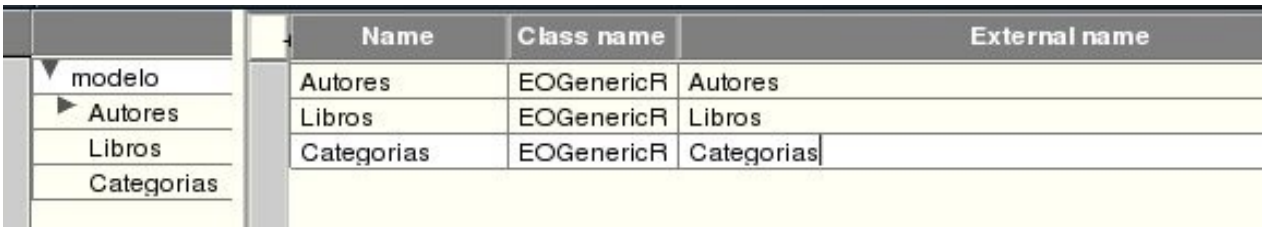
```
include $(GNUSTEP_MAKEFILES)/common.make
include $(GNUSTEP_MAKEFILES)/Auxiliary/gdl2.make
ADDITIONAL_NATIVE_LIBS+= EOControl EOAccess EOInterface

APP_NAME=Biblioteca

Biblioteca_OBJC_FILES+= \
       main.m

Biblioteca_RESOURCE_FILES= \
        Resources/modelo.eomodeld \
        Resources/Biblioteca.gorm

Biblioteca_MAIN_MODEL_FILE=Biblioteca.gorm

include $(GNUSTEP_MAKEFILES)/application.make
```

Creados estos archivos, en una terminal vamos a nuestra carpeta *Biblioteca* y escribimos **make** para compilar nuestra aplicación. Si todo sale bien, y así debe ser si se hizo todo correctamente, podremos correr nuestra aplicación con el comando **openapp ./Biblioteca.app**. El botón **Conectar** es, obviamente, para conectar con la base de datos. Esto pudo haberse hecho automáticamente agregando otro objeto a nuestra aplicación. Pero aquí sólo queremos centrarnos en el uso de GDL2. Los botones **Añadir** y **Remover**, nos permiten añadir autores o libros (antes de añadir datos debemos conectar con la base de datos). Donde los libros se añaden al autor seleccionado en la tabla Autores. Y la opción **Guardar** del menú, es para guardar los cambios realizados., de manera con los datos estén disponibles posteriormente. Y con esto tenemos finalizada nuestra primera aplicación que hace uso de GDL2.

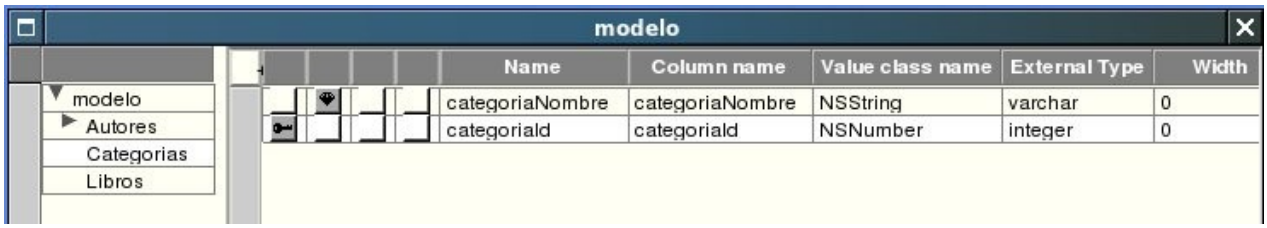


Mejorando nuestra aplicación

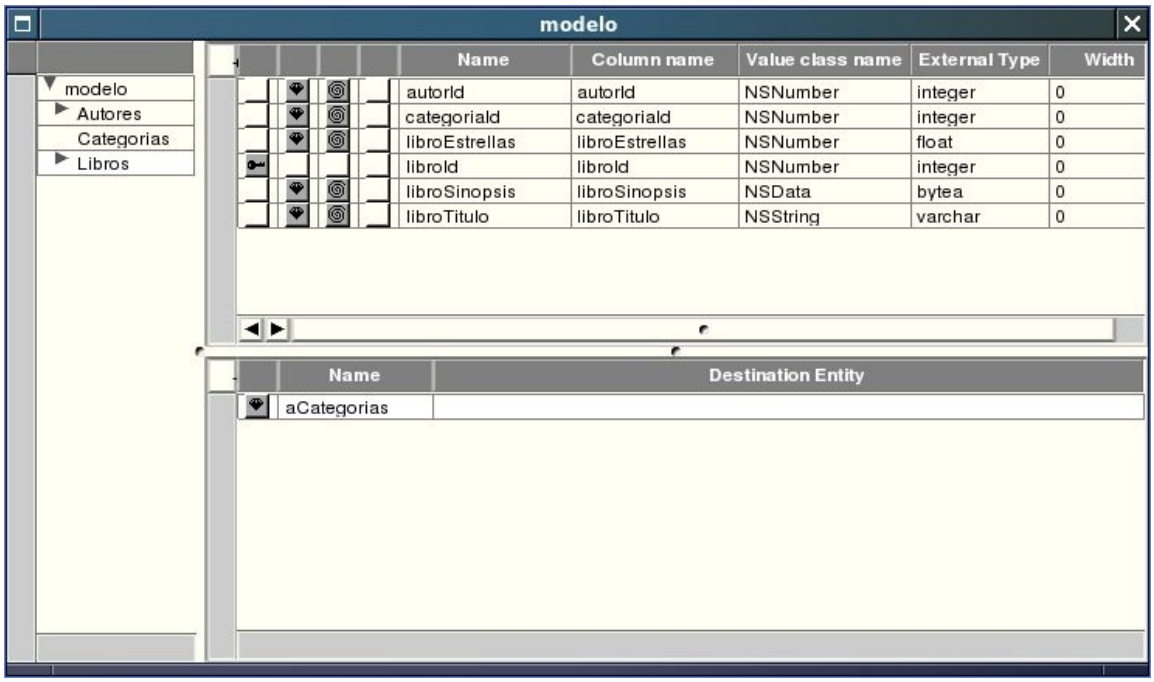
Volviendo a DBModeler, vamos a realizar algunos cambios para hacer nuestra aplicación más funcional. Primero, agregaremos una nueva entidad (tabla) a la que llamaremos **Categorias**.



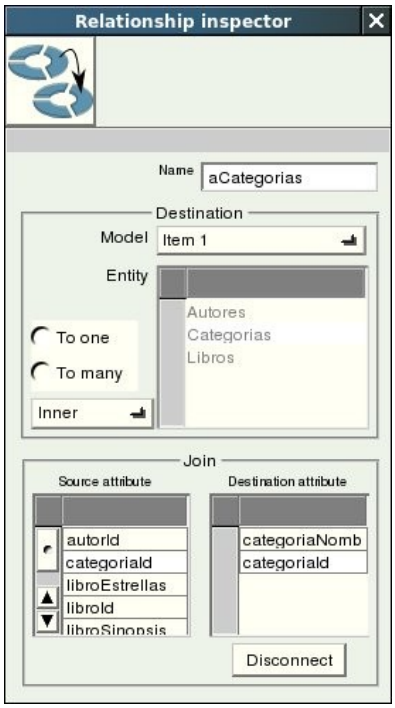
A esta nueva entidad le agregaremos dos datos, tal y como se muestra en la siguiente imagen:



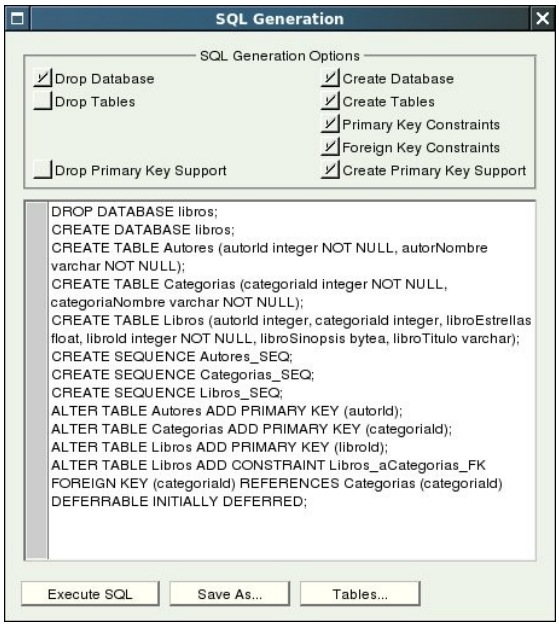
Esta nueva tabla albergara las categorías para organizar los libros (por ejemplo, Terror, Clásicos, Historia, etc.). Ahora seleccionando la tabla Libros, agregaremos tres nuevos datos, como se muestra a continuación (los datos nuevos son *categoriaId*, *libroEstrellas* y *libroSinopsis*) y una relación hacia la tabla Categorías. A la cual llamaremos *aCategorias*.



En el Inspector configuramos dicha relación como se muestra en la siguiente imagen. La relación es *uno a uno*, por lo que seleccionamos la opción **To One**. Y los datos a unir serán *categoriaId* de ambas tablas.

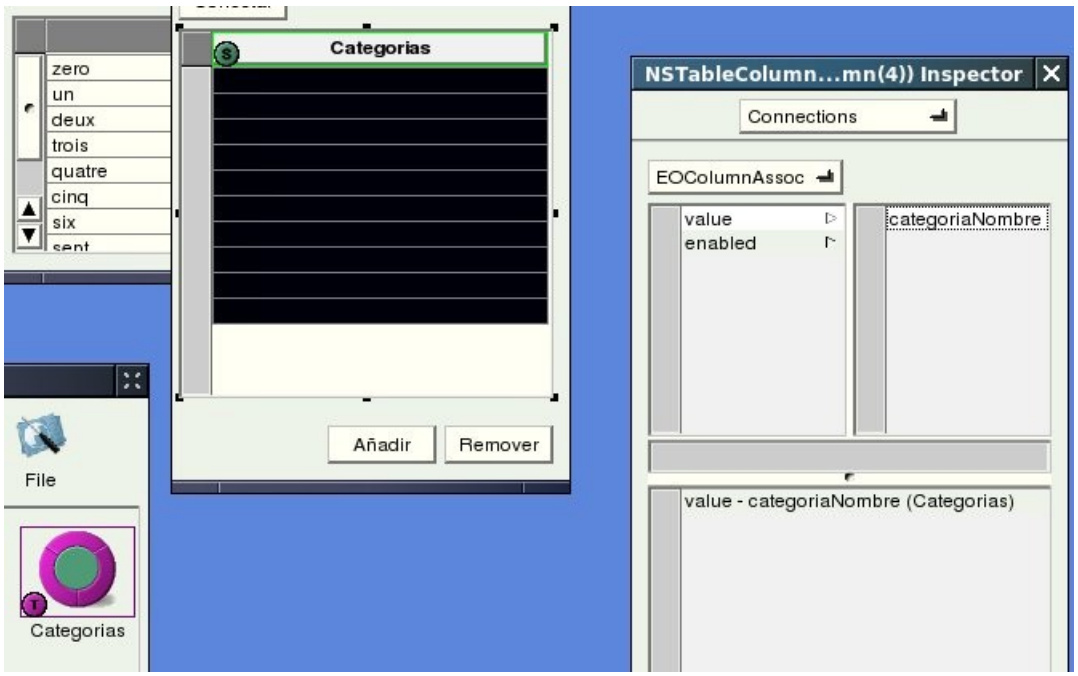


La razón de esta relación la veremos más adelante. Por el momento simplemente creemos dicha relación. Guardemos los cambios hechos al modelo y volvamos a ejecutar los correspondientes comandos SQL, seleccionando en el menú *Tools -> Generate SQL*. Si DBModeler se cerro después de terminar el ejemplo anterior, sera necesario volver a ingresar la información del conector de la base de datos (*Model -> Set Adaptor Info...*). Para este nuevo ejemplo, borro la base de datos anterior y creo una nueva a partir del nuevo modelo.

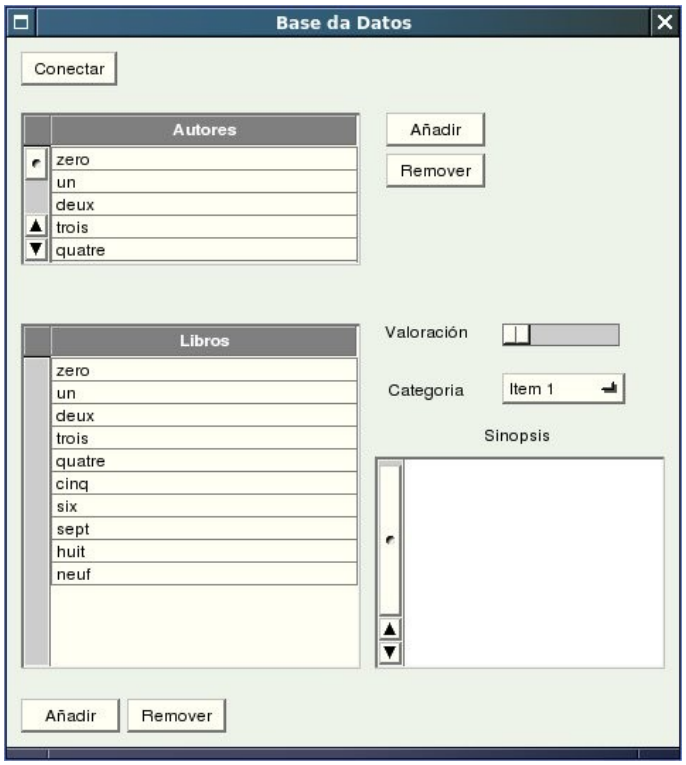


Procedemos ahora a modificar la interfaz de nuestra aplicación. Abriendo Gorm lo primero que debemos hacer, es agregar la nueva tabla **Categorías** a nuestro documento Gorm. Esto lo realizamos seleccionando dicha tabla en DBModeler y luego arrastrándola a Gorm (tal y como se hizo cuando se agrego el modelo a Gorm). Ahora vamos a agregarle una nueva ventana a nuestra interfaz. Dicha ventana la configuraremos en el Inspector para que sea visible desde el momento en que arranca la aplicación. Y debe tener el siguiente aspecto:

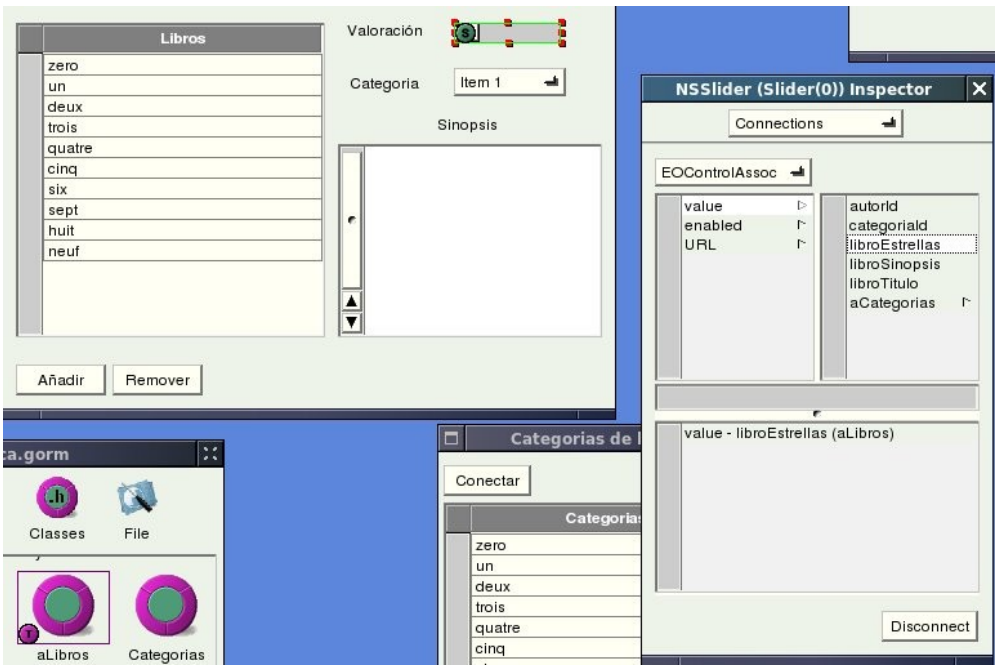
Los botones conectar, añadir y remover, se conectan como en los casos anteriores. Solo que ahora la conexión es con el objeto **Categorias**. Y la columna de la tabla se conecta de igual forma con el mismo objeto.



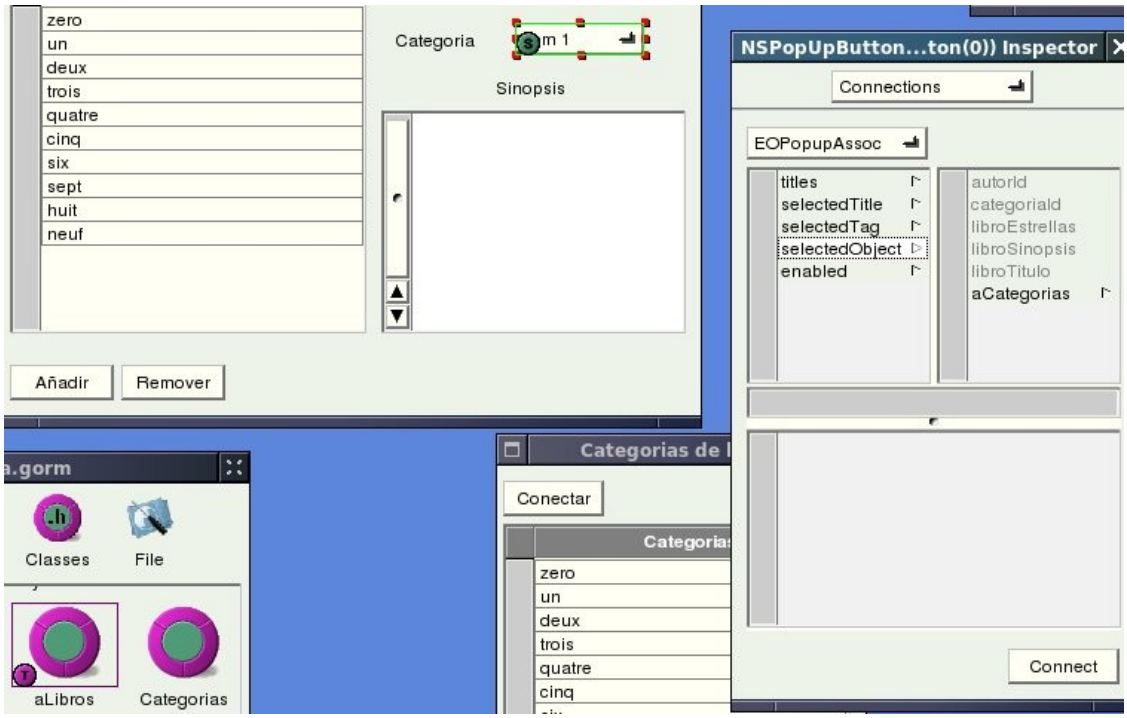
Ahora necesitamos modificar la ventana principal, para que se asemeje a la mostrada en la imagen:



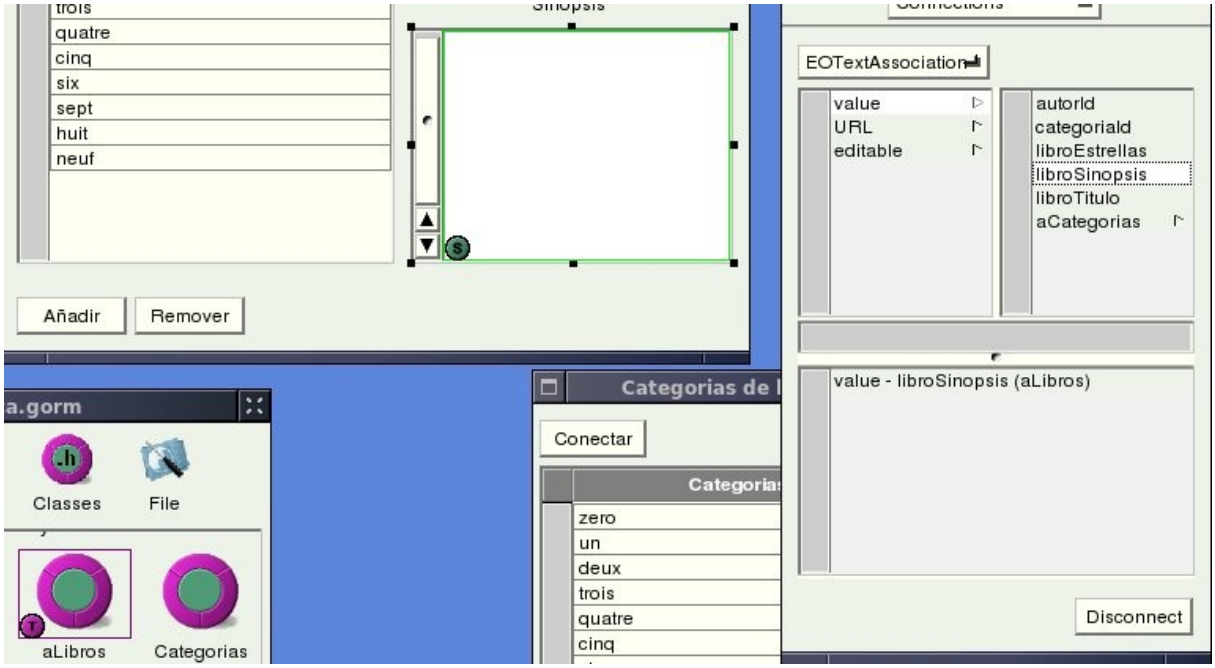
Los botones de añadir y remover de la tabla libros, los he movido a la parte inferior, y he agregado un deslizador, una lista desplegable, un visor de texto y algunas etiquetas como se ve en la imagen. Ahora procedamos a conectar estos nuevos elementos. El deslizador lo conectamos con el objeto **aLibros**. Y en el Inspector en lugar de Outlets seleccionamos **EOControlAssoc**, y dentro de *value* elegimos *libroEstrellas*.



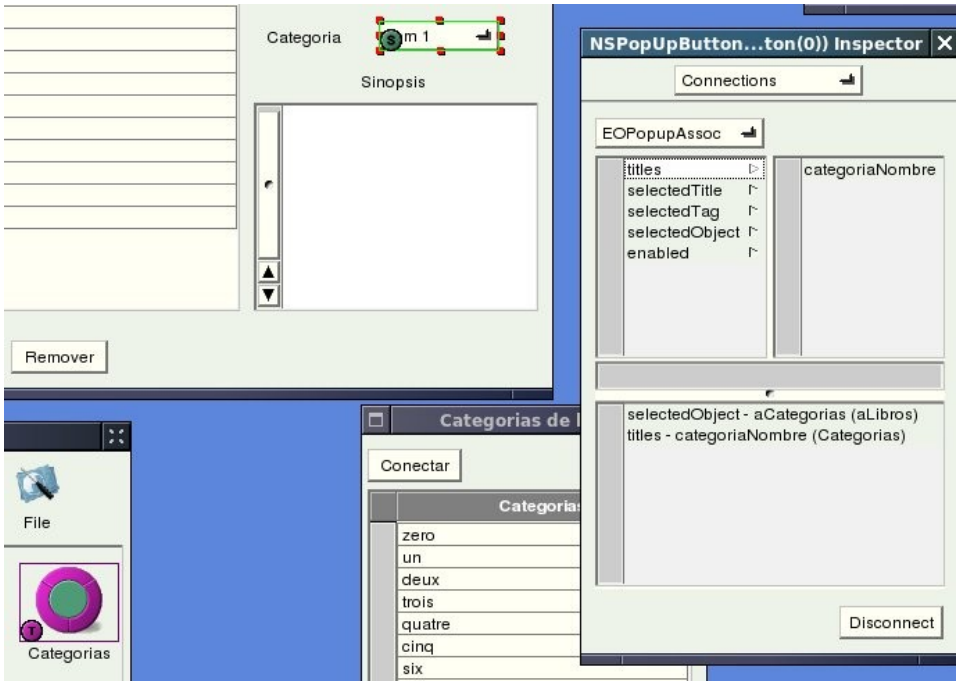
La lista desplegable la conectamos también con el objeto **aLibros**. Seleccionado **EOPopupAssoc** en lugar de Outlets, y seleccionando *aCategorias* dentro de la opción *selectedObject*. (es decir, que la opción que mostrara seleccionada la lista desplegable es la que esta almacenada en la columna *libroCategoria* de la tabla Libros).



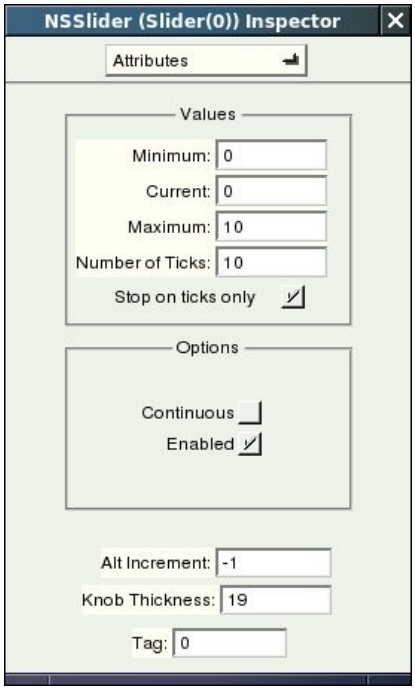
Ahora el TextView lo conectamos también con el objeto **aLibros**. Seleccionando **EOTextAssociation**, y *libroSinopsis* dentro de *value*.



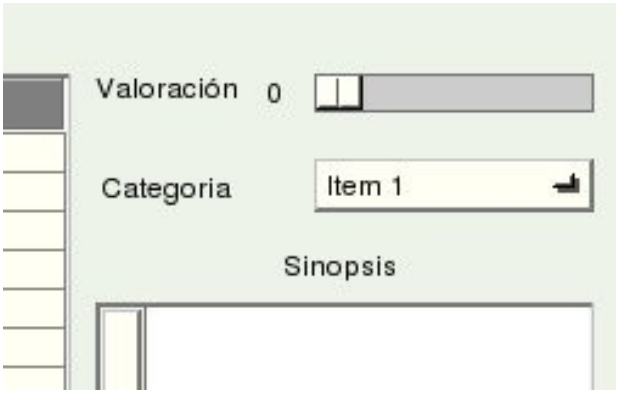
Ahora debemos conectar la lista desplegable con el objeto *Categorías* (la tabla categorías). De tal forma que al ser desplegada, muestre las categorías almacenadas en dicha tabla. Para ello seleccionamos **EOPopupAssoc**, y dentro de *titles*, seleccionamos *categoriaNombre*.



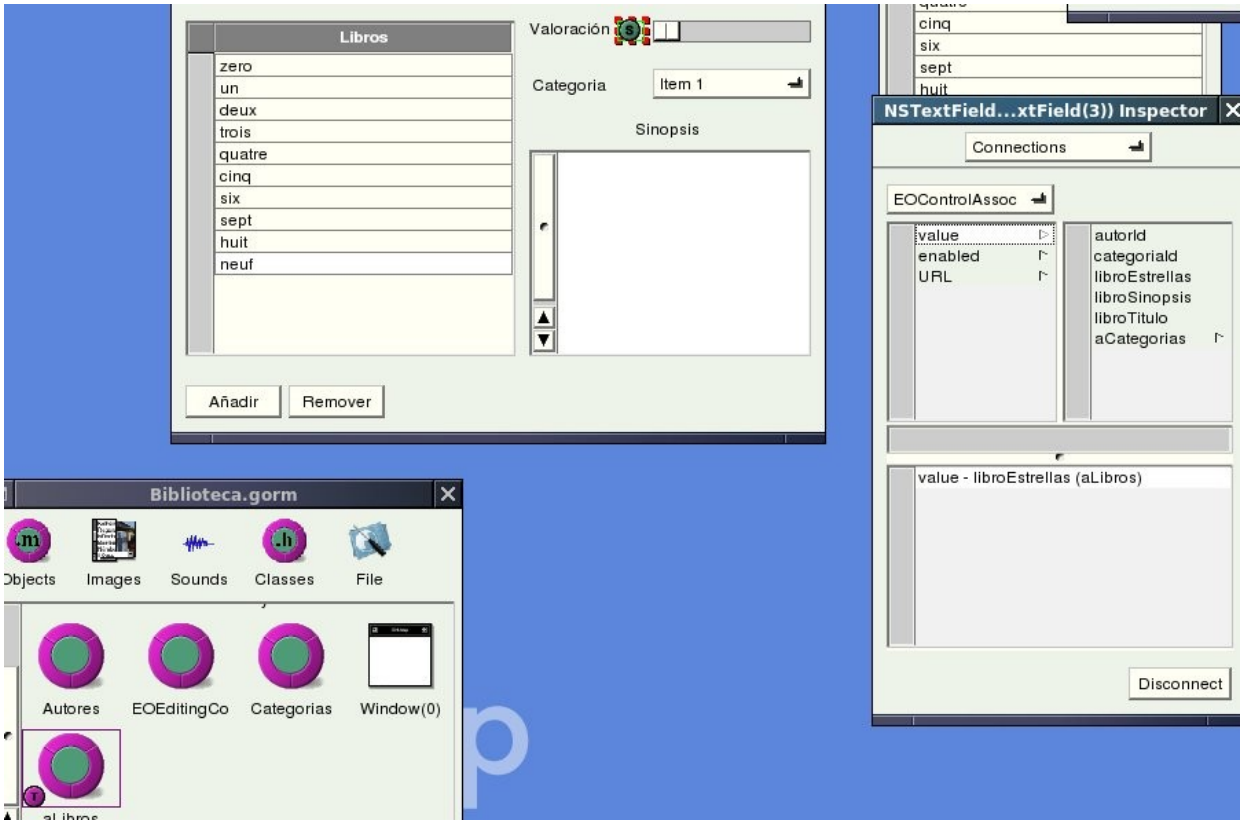
Tal y como esta nuestra aplicación en este momento, es totalmente funcional. Sin embargo, hay un problema. Si el usuario mueve el deslizador, no hay manera de que vea que valoración le ha asignado al libro. De hecho, ni siquiera hemos establecido una escala de valoración. Así que vamos ha establecer una escala de 0 a 10, para valorar los libros. Para ello, debemos configurar el deslizador como se muestra en la imagen:



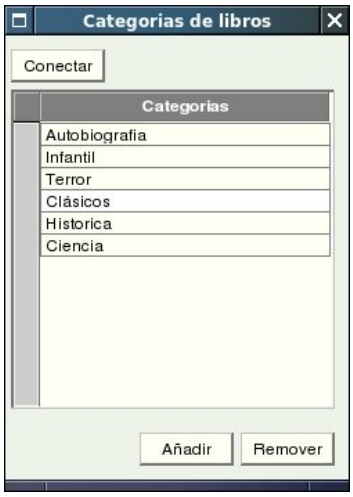
El valor mínimo es 0, el máximo es 10, el número de marcas es 10 también. El valor actual (al iniciar la interfaz) es 0. También lo he configurado para que se detenga únicamente en la marcas (es decir en valores como 1,2,3... y no en valores como 2.5, 4.87,....). Ahora, para que el usuario pueda ver que valoración ha asignado. Agregamos una etiqueta a la derecha de la que dice “Valoración”, que contendrá el texto “0”, como se muestra en la imagen:



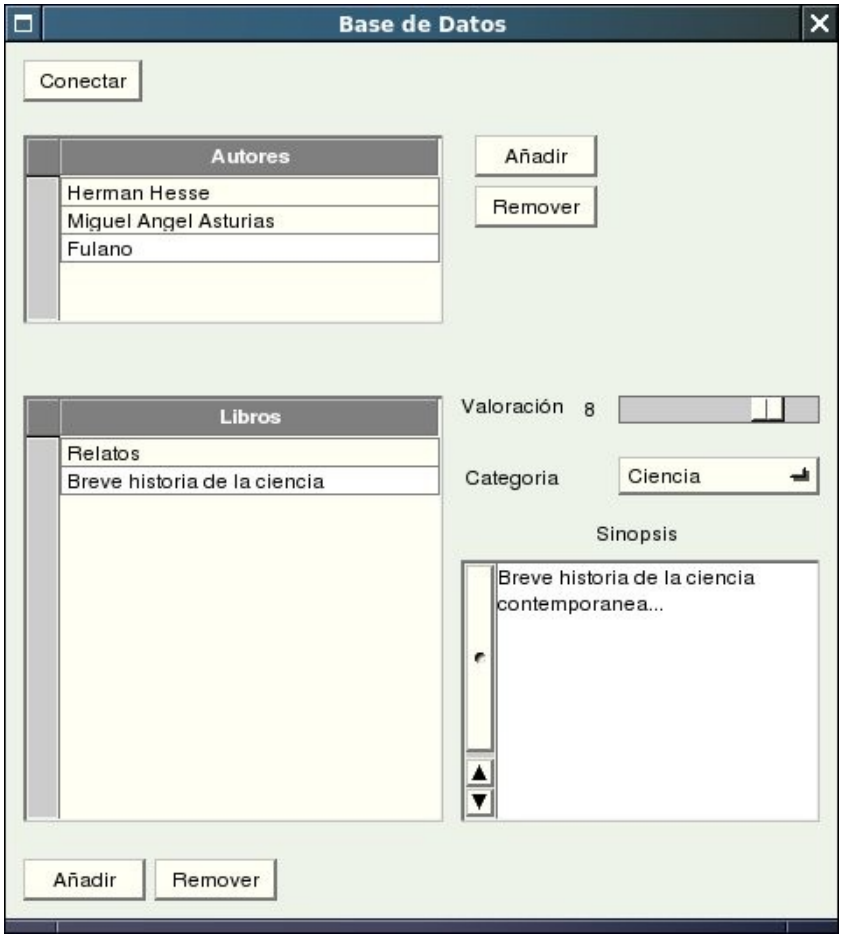
Debemos entonces conectar esta etiqueta con el objeto **aLibros**, de tal forma que muestre la valoración asignada. Esto se hace seleccionando **OEControlAssoc**, y seleccionando *libroEstrellas* dentro de *value*:



Ahora guardemos los cambios hechos en Gorm y compilemos nuevamente nuestra aplicación. Al correr nuestra aplicación, podemos primero agregar las categorías para los libros. Un clic en el botón **Conectar** y luego añadimos cuantas categorías deseemos:



Posteriormente podemos agregar los autores y sus respectivos libros, dando un clic en **Conectar** antes de ingresarlos. Sin olvidarnos de guardar los cambios en la base de datos para que estos estén disponibles posteriormente.



Como se observa de estos ejemplos, GDL2 es bastante sencillo de utilizar y facilita bastante el diseño de aplicaciones cliente-servidor. Prácticamente no se tuvo que escribir código para nuestra aplicación, exceptuando los archivos **main.m** y **GNUmakefile**.