



Categorias de libros

(GNUstep Database Library 2)

Germán Arias

enú	D Base	de Datos	Categorias
Jardar	Connector		Autobiografia
	Conectar		Infantil
cultar #h			Terror
alir #q	Autores	Añadir	Clásicos
	Herman Hesse		Historica
	Miguel Angel Asturias	Remover	Ciencia
	Fulano		
	Libros Relatos Breve historia de la ciencia	Valoración 8 Cienc	Añadir Remov
		Sinopsis Breve historia de la contemporanea	ciencia

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

Documento publicado bajo licencia GFDLv1.3

Copyright (C) 2010 German Arias.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available on <u>http://www.gnu.org/licenses/gfdl.html</u>.

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 vamos en el panel que abre, a la ruta se /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:

			_
DBMödeler	Mod	e	
Into	P New		#n
Model	In New	From Databas	ie
Edit	P Ope	า	#0
Property			
Tools	▷ Save	•	#s
Services	I> Save	e As	#8
Windows	► Reve	ert to Saved	#u
Hide	#h		
Quit	#q Set /	Adaptor Info	#1
	Swit	ch Adaptor	
	Che	ck Consistency	l

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

Model_0				×
	Name	Class name	External name	
Model_0	Entity	EOGenericR		
	Entity1	EOGenericR		

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

Model_0 Name Class name External name Model_0 Autores EOGenericR Autores	×
Model_0 Autores EOGenericR Autores	
Libros EOGenericR Libros	

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:

Model_0						×
		Name	Column name	Value class name	External Type	Width
Model_0		autorid	autorid	NSNumber	integer	0
	₽	autorNombre	autorNombre	NSString	varchar	0

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:

	Model_0								
					Name	Column name	Value class name	External Type	Width
	Model_0	-			librold	librold	NSNumber	integer	0
	Autores		-	6	autorid	autorid	NSNumber	integer	0
-	Libros		-	6	libroTitulo	libroTitulo	NSString	varchar	0

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:

Model_0							
	-		Name	Column name	Value class name	External Type	Widt
Model_0	-		autorid	autorid	NSNumber	integer	0
Autores		•	autorNombre	autorNombre	NSString	varchar	0
				22			
	-	Name		D	estination Entity		
		Relationship					

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

X	
N	ame 👝
	Relationship
D	estination
Model	tem 1 🛁
Entity I	
_	Autores
To one	Libros
10 0110	
C To many	
O To many	
To many	
To many	
To many	Join — Destination attribute
To many Inner	Join
To many Inner	- Join
To many Inner	Join Destination attribute
To many Inner	Join Destination attribute
To many Inner	Join Destination attribute

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:

-	Name	Destination Entity
	aLibros	Libros

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...*

Consistency Results	×
Success: attribute detail check Success: primary key check Success: external name check Success: relationship check Success: inheritance check	
	Cancel Ok

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)

Model_0	x
Modelo ► Autores Libros	Autores autorid autorNombre • Libros librol autorid • libroTitulo •

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:

Select adaptor
Available adaptors
PostgreSQL SQLite3
ok <

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.

PostgreSQ	L login	
Pos	stgreSQL	
Username:	german	
Password:		
Host:	[
Port:		
Database:	libros	New
	Cancel	ок <⊏

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).

SQL Generation	×						
SQL Generation Options							
⊻Drop Database	✓ Create Database						
Drop Tables	✓ Create Tables						
	Primary Key Constraints						
	✓ Foreign Key Constraints						
Drop Primary Key Support	Create Primary Key Support						
CREATE TABLE Autores (autorid integ varchar NOT NULL); CREATE TABLE Libros (libroid integer libroTitulo varchar); CREATE SEQUENCE Autores_SEQ; CREATE SEQUENCE Libros_SEQ; ALTER TABLE Autores ADD PRIMARY ALTER TABLE Libros ADD PRIMARY I	er NOT NULL, autorld integer, NOT NULL, autorld integer, ' KEY (autorld); KEY (librold);						

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).

	modelo					×
		-	Name	Column name	Value class name	External T
	♥ modelo		autorid	autorid	NSNumber	integer
	Autores		autorNombre	autorNombre	NSString	varchar
	Libros					
		Name		Destina	tion Entity	
		aLibros	Libros			
Sin nombre-1 Sin nombre-1 Objects Images Sc nds	Classes File	×				
NSOwner NE st	Ay Window NSMe	nu				
EOEditingCo Autores	aLibros	en				

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**.

Menú		
Ocultar #h	Bas	e da Datos 🛛 🗙 🗙
Salir #q	Conectar	
	Autor	es Añadir
	e zero	Bemover
	un	
	deux	
	trois	
	▼ quatre	
	Libro	us .
	zero	Añadir
	un	
	deux	Remover
	trois	
	quatre	
	cinq	
	six	

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

	L Base da L	Jatos	P !
ltar #h r #q	Zero un deux trois V quatre	Añadir Remover	Button System Bold Field: Radio Title Field: Radio Label Switch Custor
	Zero un deux trois quatre cinq six sent	Añadir Remover	NSButton (Button(0)) Inspect Connections
a.gorm Classes	File My Window		delete: editingCon enterQuery ▲ fetch: insert: fetch: (Autores)

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:

autorNombre aLibros ⊵	autorld libroTitulo
►	
parent - aLibros (Auto	ores)

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

GNUstep Database Library 2

www.gnustep.wordpress.com

Gultar #h Salir #q	Base da Datos Conectar	Añadir Remover	Text Item 1 Button System Bold Badio System Radio Title Radio Label Box Fixed Switch T CustomView
	Zero un deux trois quatre cinq six sent	Añadir Remover	NSMenuitemuitem(3)) Inspector Connections
Biblioteca.gorm Sounds Classes	File Autores		noop: _invalidateObject\ _processObjectStr forgetObject: saveChanges: refetch: _objectsInitialized

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:



Al segundo archivo lo llamaremos GNUmakefile y contendrá lo siguiente:

U.D 4 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.

Menú		
Guardar	Base da Datos	>
Ocultar #h		
Salir #q		
	Autores	Añadir
	Herman Hesse	Bemover
	Miguel Angel Asturias	
	Salome Gil	
	Libros	
	Bajo la rueda	Añadir
	El lobo estepario	
		Remover

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**.

	Name	Class name	External name
modelo	Autores	EOGenericR	Autores
Autores	Libros	EOGenericR	Libros
Libros	Categorias	EOGenericR	Categorias
Categorias		- 68	1. 83739

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

	modelo							
		Name	Column name	Value class name	External Type	Width		
▼ modelo		categoriaNombre	categoriaNombre	NSString	varchar	0		
Autores		categoriald	categoriald	NSNumber	integer	0		
Categorias								
Libros								

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*.

	modelo							
	4			Name	Column name	Value class name	External Type	Width
▼ modelo		• 6		autorld	autorid	NSNumber	integer	0
Autores		• 6		categoriald	categoriald	NSNumber	integer	0
Categorias		• 6		libroEstrellas	libroEstrellas	NSNumber	float	0
Libros				librold	librold	NSNumber	integer	0
		• 6		libroSinopsis	libroSinopsis	NSData	bytea	0
	1000	• 6		libroTitulo	libroTitulo	NSString	varchar	0
		-) Na	ame		De	estination Entity		
	-	aCateg	orias					

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 *categoriald* de ambas tablas.

Relation	ship inspector	×
50		
	Name aCategorias	
Model	Item 1	-1
Entity		
To one To many	Autores Categorias Libros	
Inner 🛁		
Source attribute	Join — Destination attribute	•
autorid categoriald libroEstrella	categoriaNor categoriald	nb
librold	ie	
	Disconnect	

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.

3QL 08	meration
SQL Gener	ation Options
✓ Drop Database	⊻ Create Database
Drop Tables	Create Tables
	Primary Key Constraints
	Foreign Key Constraints
Drop Primary Key Support	Create Primary Key Support
DROP DATABASE libros; CREATE DATABASE libros; CREATE TABLEA utores (autorid varchar NOT NULL); CREATE TABLE Categorias (cate categoriaNombre varchar NOT NL CREATE TABLE Categorias (cate categoriaNombre varchar NOT NL CREATE SEQUENCE (autorid in float, libroid integer NOT NULL, lit CREATE SEQUENCE Autores_St CREATE SEQUENCE Libros_SEC ALTER TABLE Autores ADD PRIM ALTER TABLE Autores ADD PRIM ALTER TABLE Libros ADD PRIM ALTER TABLE Libros ADD PRIM ALTER TABLE Libros ADD CONS FOREIGN KEY (categoriad) PEFP DEFERRABLE INITIALLY DEFER	integer NOT NULL, autorNombre goriald integer NOT NULL, JLL); iteger, categoriald integer, libroEstrellas iroSinopsis bytea, libroTitulo varchar); EQ; SEQ; Q; ARY KEY (autorld); RIMARY KEY (autorld); ARY KEY (labrold); ARY KEY (librold); ITRAINT Libros_aCategorias_FK ERENCES Categorias (categoriald) RED;

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:

	Categorias de libros	×
0	Conectar	
	Categorias	
Γ	zero	
	un	
	deux	
	trois	
	quatre	
	cinq	
	six	
	sept	
	huit	
	neuf	
		_
	Añadir Remove	r

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.

		(s) Categorias	-		
	zero		NSTab	leColumnn	nn(4)) Inspector X
	un				
	deux			Connections	
	trois				
	quatre		EOCo	umnAssoc 🛥	
	cinq				
HŞ.	six	•	val	ue 🗅	categoriaNombre
Ľ	ll sent		ena	abled 1~	
Ø		Añadir Remover			
F	le		- I wat	e enterenie Ne	mbus (Ostanavias)
	Categorias		Vai	ue - calegonalivo	indre (Calegonas)

	Autores	Añadir
T	zero	
ł	un	Remover
ľ	deux	
I	trois	
İ	quatre	
ŀ	un deux	Categoria Item 1 🚽
	un	Categoria Item 1 🛁
ŀ	deux	Sinonsis
ŀ	quatra	
ŀ	cing	
ŀ	six	
ŀ	sept	
ľ	huit	
ľ	neuf	
12-		
Ľ		

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*.

Libros	Valoración 🛐	
zero		
un	Categoria Item 1 -	NSSlider (Slider(0)) Inspector
deux		
trois	Sinopsis	Connections -
quatre		
cinq		EOControlAssoc 🚽
six		
sept		value D autorid
huit		enabled in categoriaid
neuf		UNL IIDROEStrellas
		libroSinopsis
		acategorias
Añadir Bemover		
	Categorias d	e value - libroEstrellas (aLibros)
orm 🛛 🕄		
	Conectar	
(.h) (
Sec. 19	Catego	ria:
lasses File	7010	
	2210	
	dour	
	deux	-
	trois	Dissepaget
	quatre	Disconnect

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).

zero		
un	Categoria 😗 1 🚽	NSPopUpButtonton(0)) Inspector >
deux		
trois	Sinopsis	Connections 🚽
quatre		
cinq		EOPopupAssoc 🚽
six		
sept		titles r autorid
huit		selected little I* categoriald
neuf		selected lag P libroEstrellas
		selectedObject P libroSinopsis
Añadir Remover		-
orm 🔀	🗖 Categorias de	
🕒 🐼 📕	Conectar	
lasses File		
	Zero	
	deux	-
	trois	
	quatre	Connect
	cing	
aLibros Categorias		

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

trois	onopsis	
quatre		
cinq		EOTextAssociation=
six		
sept		value ▷ autorid
huit		URL ^{r-} categoriald
neuf		editable 1 ⁻ libroEstrellas
		libroSinopsis
Añadir Remover		libroTitulo aCategorias r
		value - libro Sinoneie (al ibros)
a.gorm 🔀	Categorias de	value - infoomopais (actions)
(1) (3)	Conectar	
Classes File	7810	
*		
	deux	
	trois	
	quatre	Disconnect
al ibras Catagorias	cing	
acibios Galegonas		

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:

NSSlider (Slider(0)) Inspector	×
Attributes 🛁	
Values	
Minimum: 0	
Current: 0	
Maximum: 10	
Number of Ticks: 10	
Stop on ticks only 🗾	
Options	
Continuous Enabled <u>//</u>	
Alt Increment: -1 Knob Thickness: 19 Tag: 0	

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:

Valoración 0	
Categoria	ltem 1 🛁
-	Sinopsis

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*:

	Libros	Valoración 🚯	cinq
	zero	Categoria Item 1 🚽	six sept huit
	deux trois quatre	Sinopsis	NSTextFieldxtField(3)) Inspector X Connections
	six sept huit neuf		EOControlAssoc
l Bil	Añadir Remover		value - libroEstrellas (aLibros)
Autores EOEc	itingCo Categorias Window(0)	n	Disconnect
		Υ	

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:

Conectar Categorias Autobiografia Infantil Terror Clásicos Historica Ciencia Añadir Remover		Categorias de libros 🛛 🗙
Categorias Autobiografia Infantil Terror Clásicos Historica Ciencia	c	onectar
Autobiografia Infantil Terror Clásicos Historica Ciencia Añadir Remover		Categorias
Infantil Terror Clásicos Historica Ciencia Añadir Remover		Autobiografia
Terror Clásicos Historica Ciencia Añadir Remover		Infantil
Clásicos Historica Ciencia Añadir Remover		Terror
Añadir Remover		Clásicos
Ciencia Añadir Remover		Historica
Añadir Remover		Ciencia
Añadir Remover		
Añadir Remover	-	
		Añadir Remover
	1	

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.

□ Base de	Datos	×
Conectar	r ()	
Autores Herman Hesse Miguel Angel Asturias Fulano	Añadir Remover	
Libros	Valoración 8	
Breve historia de la ciencia	Categoria Ciencia -	a i
	Sinopsis	
	Breve historia de la ciencia contemporanea	
Añadir Remover		

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**.