TUTORIAL DE

cakePHP

ÍNDICE

1. INTRODUCCIÓN

CakePHP es un framework para programar aplicaciones Web que sigue la arquitectura MVC (Modelo Vista Controlador: http://es.wikipedia.org/wiki/MVC). Para su funcionamiento requiere un servidor Web Apache, con PHP (versión 4 o 5) y un servidor de base de datos MySQL (aunque también trabaja con '*PostgreSQL*', '*SQLite*' o '*ADODB*'). Existen herramientas, como XAMPP (http://www.apachefriends.org/en/xampp.html), que integran ambos servicios en una única instalación. El presente tutorial se ha realizado con la versión 1.6.0 de XAMPP, que incorpora las versiones 2.2.4 de Apache, 5.2.1 y 4.4.5 de PHP y 5.0.3.3 de MySQL. La versión de cakePHP con la que se ha trabajado y cuyo funcionamiento se va a describir es la 1.1.14.4797 de abril de 2007. Esta herramienta se puede descargar de su web oficial:

http://www.cakephp.org/

Los lectores de este tutorial deberán tener conocimientos de programación en PHP, de gestión de MySQL y conceptos teóricos de bases de datos.

2. CONFIGURACIÓN

Una vez instalado un servido (Apache), con el módulo '*rewrite*' habilitado (aunque también funciona sin él, pero es preferible según el autor del framework). Dicha habilitación se realiza accediendo al fichero '*httpd.conf*' en la carpeta '*[…] /apache/conf/*' y descomentando la línea correspondiente (quitando la '#') antes de arrancar el servicio web.

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

Figura 1. Activación del módulo 'rewrite' de Apache

Se crea una carpeta dentro de la web de nuestro servidor en la que descomprimir los ficheros del cakePHP. En nuestro caso, dicha carpeta se llamará '*proyecto_cake*', y accederemos a ella vía web mediante la dirección '*http://localhost/proyecto_cake*' (ver figura 2).

Lo siguiente que hay que hacer es configurar el acceso a la base de datos; para ello se crea un archivo con el nombre '*database.php*' en la carpeta '*[…] /proyecto_cake/app/config/*', donde se especifican los parámetros de la conexión: el motor de base de datos, el tipo de conexión ('*mysql_connect*' o '*mysql_pconnect*'), el host, el login, la contraseña de la base de datos y el nombre de ésta, que en este tutorial será '*proyecto_cake*'; naturalmente esta base de datos deberá estar creada de antemano (ver figura 3).

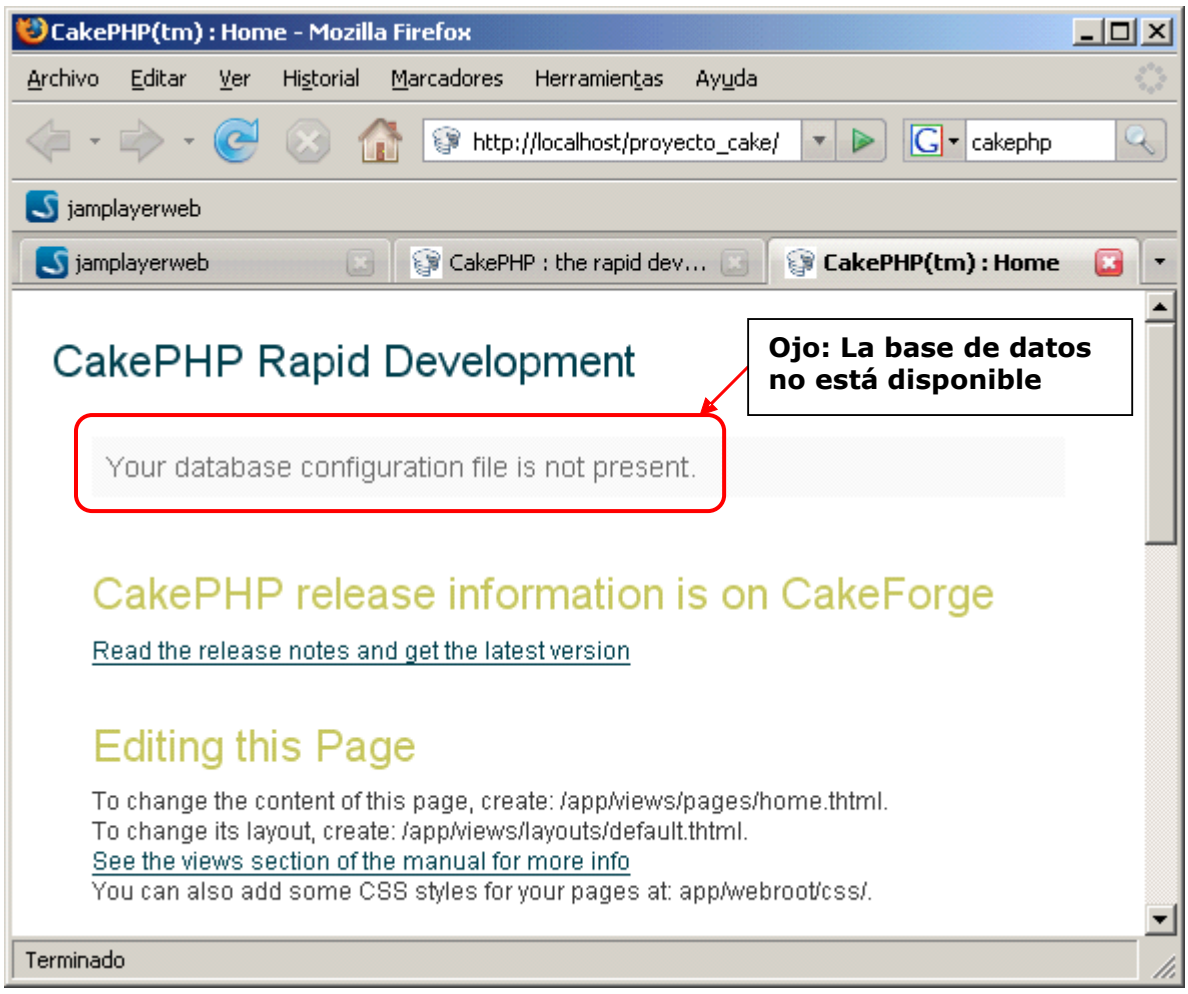


Figura 2. Vista inicial CakePHP

```
<?php
class DATABASE_CONFIG{
```

```
    var $default =
         array('driver' => 'mysql',
               'connect' => 'mysql_connect',
               'host' => 'localhost',
               'login' => 'user',
               'password' => 'pwd_user',
               'database' => 'proyecto_cake',
               'prefix' => ''); /*este parámetro se queda en blanco*/
}?>
```

Figura 3. Configuración de acceso a la base de datos

Ahora CakePHP ya puede acceder a la base de datos (ver figura 4).

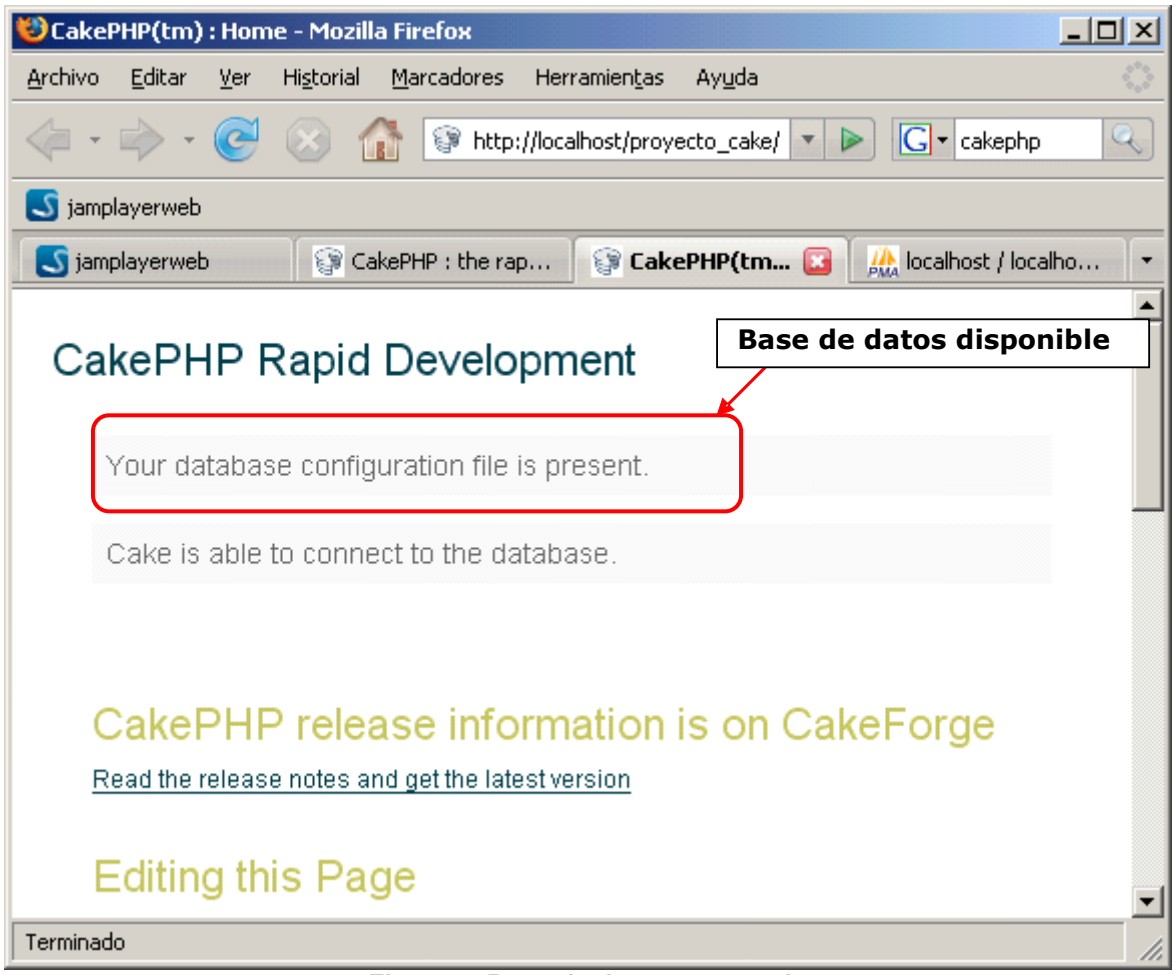


Figura 4. Base de datos conectada

3. CONCEPTOS SOBRE CakePHP

Los diferentes elementos necesarios para desarrollar una aplicación con CakePHP deben ser nombrados siguiendo el siguiente convenio:

- Tablas de la base de datos.- El nombre de cualquier tabla debe estar en plural, ej: '*proyectos*' o '*alumnos*'. En el caso de ser una tabla para una relación muchos a muchos, por ejemplo entre 'proyectos' y 'alumnos', se crearía otra tabla cuyo nombre contendría los de las tablas relacionadas separadas por una barra ('_') y en orden alfabético, es decir: '*alumnos_proyectos*' (ojo: si la tabla se llamara '*proyectos_alumnos*' no funcionaría).Ver ejemplo en la figura 5.
- Model (modelo).- Cada tabla debe tener una clase modelo de igual nombre que la tabla, pero en singular, guardada en un archivo también con el mismo nombre de la tabla, pero en singular, con extensión '.php'. Dicho fichero deberá almacenarse en la carpeta '*[…]/proyecto_cake/app/models/*'. Para la tabla 'alumnos', la clase para el modelo se llamará '*class alumno… { … }*' y deberá estar guardada en el fichero '*alumno.php*'.

```
CREATE TABLE `alumnos` (
   `id` int(11) NOT NULL auto_increment,
   `nombre` varchar(50) NOT NULL,
   `apellido` varchar(50) NOT NULL,
   `fecha_nacimiento` date NOT NULL,
   `email` varchar(45) NOT NULL,
   `telefono` varchar(15) NOT NULL,
   `num_natricula` int(4) NOT NULL,
   `nota_selectividad`decimal(4,2) NOT NULL,
   PRIMARY KEY (`id`)
);

CREATE TABLE `proyectos` (
   `id` int(10) unsigned NOT NULL auto_increment,
   `titulo` varchar(45) NOT NULL,
   `num_paginas` varchar(45) NOT NULL,
   PRIMARY KEY (`id`)
);

CREATE TABLE `alumnos_proyectos` (
   `id` int(11) NOT NULL auto_increment,
   `alumno_id` int(11) NOT NULL,
   `proyecto_id` int(11) NOT NULL,
   PRIMARY KEY  (`id`)
);
```

Figura 5. Ejemplo creación de tablas

```
<?php
class Alumno extends AppModel{
   var $name = 'Alumno';
}
?>
```

Figura 6. Creación del modelo alumno.php

- Controller (controlador).- El nombre de la clase controlador se forma poniendo el nombre de la tabla en plural, seguido de '*Controller*'. Para la

tabla 'alumno' quedaría así: '*class alumnosController… { … }*'. Esta clase deberá guardarse en un fichero de nombre '*alumnos_controller.php*' dentro de la carpeta '*[…]/proyecto_cake/app/controllers/*'.

```
<?php
class alumnoscontroller extends AppController{
   var $name = 'alumnos';
   var $helpers = array('Html', 'Form');
}
?>
```

Figura 7. Creación del controlador alumno.php

- Views (vistas).- Para crear las vistas de las tablas de la base de datos, hay que crear una carpeta por cada tabla dentro de la carpeta '*[…] /proyecto_cake/app/views/*'. Para la tabla 'alumnos' se creará la carpeta '*[…]/proyecto_cake/app/views/alumnos/*'. Dentro de cada una de las carpetas creadas para las vistas se creará un fichero para la vista de edición, de nombre '*edit.thtml*', otro para la de inserción, '*add.thtml*', otro para el listado de registros '*index.thtml*' y otro para cada registro '*view.thtml*'.

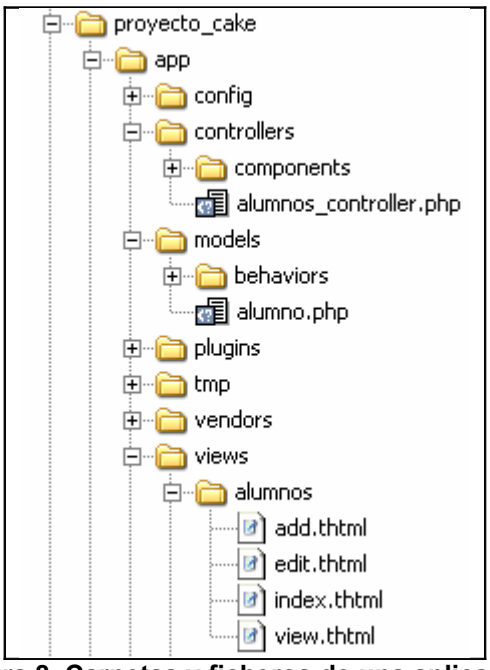


Figura 8. Carpetas y ficheros de una aplicación

CakePHP dispone de la clase '*HtmlHelper*' que permite crear formularios en HTML y algunos otros componentes para una página web, es decir, para crear las vistas

de la aplicación (ficheros con extensión '*.thtml*'). En los archivos de vistas está disponible el objeto '*html*' de la clase '*HtmlHelper*' que proporciona los métodos necesarios para diseñar cada vista, algunos de los más utilizados son:

- Para crear el formulario:

```
HtmlHelper::formTag($target=null,$type='post',$htmlAttributes=array())
```

- Para enviar la información de un formulario:

```
HtmlHelper::submit($caption='Submit',$htmlAttributes=array(),$return=false)
```

- Para que el campo password, convierta la contraseña a asteriscos:

```
HtmlHelper::password($fieldName,$htmlAttributes=array(),$return=false)
```

- Agrega los campos para textos de mayor longitud:

```
HtmlHelper::textarea($fieldName,$htmlAttributes=array(),$return=false)
```

- Crea campos ocultos:

```
HtmlHelper::hidden($fieldName,$htmlAttributes=array(),$return=false)
```

- Agrega imágenes a los formularios:

```
HtmlHelper::image($path,$htmlAttributes=array(),$return=false)
```

- Crea campos para texto:

```
HtmlHelper::input($fieldName,$htmlAttributes=array(),$return=false)
```

- Crea campos para seleccionar elementos:

```
HtmlHelper::selectTag($fieldName,$optionElements,$selected=null,
                      $selectAttr=array(),$optionAttr=null,
                      $showEmpty=true,$return=false)
```

- Genera campos para insertar la fecha y hora:

```
HtmlHelper::dateTimeOptionTag(
   $tagName, $dateFormat='DMY', $timeFormat='12', $selected=null,
   $selectAttr=null, $optionAttr=null, $showEmpty=true)
```

Toda la documentación correspondiente a esta versión se puede encontrar en http://api.cakephp.org/.

A continuación se va a crear un ejemplo con un formulario para la tabla alumno (siguiendo con el ejemplo anterior). Este formulario se hará para crear un alumno nuevo, por tanto tendremos que ir a la carpeta view, '*[…]/proyecto_cake/app/views/*', y crear un archivo llamado add.thtml. Como el que se puede ver en la figura 9.

Además de estas funciones, CakePHP, posee la característica de validar los datos, pudiéndolo hacer para cuatro tipos de campos, que son los siguientes:

- Campo no vacío: VALID_NOT_EMPTY.
- Campo numérico: VALID_NUMBER.
- Campo de e-mails: VALID_EMAIL.
- Campo de años: VALID_YEAR.

```
<br><h3>Nuevo Alumno</h3>
<?php echo $html->formTag('/alumnos/add/'));?>
<table>
<div class="required">
  //Indicar si es un campo opcional(optional) u obligatorio(required)
  //indica el nombre del campo en el formulario.
  <label for="alumno_nombre">Nombre:</label>
  //Primero poner el tipo de campo(input), seguido del nombre de la tabla/campo
  <?php echo $html->input('Alumno/nombre', array('size' => '50'));?>
  <?php echo $html->tagErrorMsg('Alumno/nombre', 'Nombre no válido');?>
</div>

<div class="required">
  <label for=" alumno _apellido">Apellido:</label>
  <?php echo $html->input(' Alumno /apellido', array('size' => '50'));?>
  <?php echo $html->tagErrorMsg('Alumno/apellido', 'Apellido no válido');?>
</div>

<div class="optional">
  <label for=" alumno_fecha_nacimiento">Fecha de Nacimiento:</label>
  <?php echo $html->dateTimeOptionTag('Alumno/fecha_nacimiento','DMY','NONE');?>
  <?php echo $html->tagErrorMsg('Autor/fecha_nacimiento','Fecha no valida');?>
</div>

<div class="required">
  <label for="alumno_email">Email:</label>
  <?php echo $html->input('Alumno/email', array('size' => 50));?>
  <?php echo $html->tagErrorMsg('Alumno/email', 'E-mail no valido');?>
</div>

<div class="optional">
  <label for=" alumno_telefono">Telefono:</label>
  <?php echo $html->input('Alumno/telefono',array('size' => 50));?>
  <?php echo $html->tagErrorMsg('Alumno/telefono','Deben ser numeros');?>
</div>

<div class="required">
  <label for=" alumno_num_matricula">Número de matricula:</label>
  <?php echo $html->input('Alumno/num_matricula', array('size'=>50));?>
  <?php echo $html->tagErrorMsg('Alumno/num_matricula',
                                'Debe escribir un año');?>
</div>

<div class="required">
  <label for=" alumno_nota_selectividad">Nota de Selectividad:</label>
  <?php echo $html->input('Alumno/nota_selectividad',array('size'=>50));?>
  <?php echo $html->tagErrorMsg('Alumno/nota_selectividad',
                                'Deben ser un número');?>
</div>

</table>
```

```
<div class="submit"><input type="submit" value="Add" /></div>
</form>
```

Figura 9. Formulario para crear nuevos alumnos

A continuación se van a validar los datos de la tabla alumnos, siguiendo el siguiente criterio.

- Campos no vacíos: nombre, apellido.
- Campo email: email.
- Campo año: año
- Campo numero: nota_matricula

Para hacer esta validación en cake, tendremos que ir a *'[…] /proyecto_cake/app/views/alumnos/'* y crear las validaciones como se puede ver en la figura 10.

```
<?php
class alumno extends AppModel{
  var $name = 'Alumno';
  var $validate = array('nombre'=>VALID_NOT_EMPTY,
                        'apellido'=>VALID_NOT_EMPTY,
                        'email'=>VALID_EMAIL,
                        'año_matriculación'=>VALID_YEAR,
                        'nota_selectividad'=>VALID_NUMBER,);
}
?>
```

Figura 10. Validar campos del formulario

Una opción muy potente de la que dispone CakePHP, es la gestión de las relaciones entre tablas de la base de datos. Estas relaciones se deben escribir en el modelo de la tabla correspondiente (alumnos.php, proyecto.php). A continuación se van a ver los cuatro tipos de relaciones que tiene CakePHP, viendo para cada una un ejemplo entre las tablas “alumnos” y ”proyectos”:

- hasOne: 1:1, uno a uno. Siguiendo el ejemplo, se dirá que la relación entre proyecto y alumno es de uno a uno, por tanto en uno de los modelos (alumno.php o proyecto.php), se debe incluir esta condición, es decir habrá que validar la relación entre tablas para que sea de uno a uno. Veasé figura 11.

```
<?php class Alumno extends AppModel{
   var $name = 'Alumno';
   var $hasOne = array('Profile' => array('className' => 'Proyecto',
                'conditions'=> '', 'order'     => '',
                'dependent' =>true, 'foreignKey'=> 'alumno_id'));
}?>
```

Figura 11. Validar relación uno a uno

'className' (obligatorio): nombre del modelo que se quiere asociar. En este ejemplo se quiere asociar con el modelo Proyecto.

'conditions': condiciones SQL que definen la relación. Se podría utilizar, por ejemplo, para asociar solo los proyectos que tienen 200 páginas, para especificarlo se pondría: *"Proyecto.num_paginas='200'"*.

'order': se utilizaría para ordenar la asociación en el modelo. Para ordenarlo de forma ascendente se pondría: *"Proyecto.titulo ASC"*. Y para ordenarlo de forma descendnte se pondría: *"Proyecto.titulo DESC"*

'dependent': si se pone a "true", el registro que esta asociado a otro registro se borrará cuando este se borré. Por ejemplo si el alumno "Carlos" esta asociado con el proyecto "Aplicaciones Web", si se borra el alumno "Carlos" también se borrará el proyecto "Aplicaciones web".

'foreignKey': es el nombre de la FK que apunta al modelo con el que está asociado. En este caso en la tabla proyecto debe haber un campo que sea alumno_id, pues este campo es la FK que tendremos que escribir en "foreign key".

```
Array(
```

```
    [Alumno] => Array(
                    [id] => 25
                    [nombre] => Carlos
                    [apellido] => Lirón López
                    [fecha_nacimiento] => 1982-11-11
                    [email] => carlosliron@alu.umh.es
                    [telefono] => 965350895
                    [año_matriculacion] => 2003
                    [nota_selectividad] => 9,8
                )
    [Proyecto] => Array(
                    [id] => 4
                    [titulo] => Aplicaciones Web
                    [num_paginas] => 200
                    [alumno_id] = 25
                )
)
```

Figura 12. Resultado de la relación 1:1 en Alumno y Proyecto

- hasMany: 1:N, uno a muchos. Siguiendo el ejemplo, se dirá que la relación entre alumno y proyecto es de uno a muchos, por tanto en el modelo alumno.php, se debe incluir esta condición, es decir habrá que validar la relación entre tablas para que sea de uno a muchos. Véase figura 13.

  'className' (obligado): nombre del modelo con el que se quiere asociar. En este caso con 'Proyecto'.

  'conditions': condiciones SQL que definen la relación.

  order': se utilizaría para ordenar la asociación en el modelo.

  'limit': el número máximo de asociaciones que se quieren en un modelo. En este ejemplo solo se quieren dos proyectos por cada alumno.

```
<?php
class Alumno extends AppModel{
   var $name = 'Alumno';
   var $hasMany = array('Proyecto' =>
                           array('className'  => 'Proyecto',
                                 'conditions' => '',
                                 'order'      => '',
                                 'limit'      => '2',
                                 'foreignKey' => 'alumno_id',
                                 'dependent'  => true,
                                 'exclusive'  => false,
```

```
                                    'finderQuery'=> ''
                                )
                    );
}
?>
```

Figura 13. Validar relación uno a muchos

'foreignKey': es el nombre de la FK que apunta al modelo con el que está asociado. En este caso en la tabla proyecto debe haber un campo que sea alumno_id, pues este campo es la FK que tendremos que escribir en "foreign key".

'dependent': si se pone a "true", el registro que esta asociado a otro registro se borrará cuando este se borré. Por ejemplo si el alumno "Jose" esta asociado con el proyecto "Ajax", si se borra el alumno "Jose" también se borrará el proyecto "Ajax".

'exclusive': si se pone a true, todos los objetos asociados serán borrados en una consulta SQL.

'finderQuery': especifica una sentencia completa en SQL para buscar asociaciones.

```
Array(
   [Alumno] => Array(
                  [id] => 25
                  [nombre] => Carlos
                  [apellido] => Lirón López
                  [fecha_nacimiento] => 1982-11-11
                  [email] => carlosliron@alu.umh.es
                  [telefono] => 965350895
                  [año_matriculacion] => 2003
                  [nota_selectividad] => 9,8
               )
   [Proyecto] => Array(
                    [0] => Array(
                              [id] => 4
                              [titulo] => Aplicaciones Web
                              [num_paginas] => 200
                              [alumno_id] = 25
                           )
                    [1] => Array(
                              [id] => 20
                              [titulo] => Bases de Datos
                              [num_paginas] => 150
                              [alumno_id] = 25
```

```
                                )
                        )
)
```

Figura 14. Resultado de la relación 1:N en Alumno y Proyecto

- belongsTo: N:1, muchos a uno. Siguiendo el ejemplo, se dirá que la relación entre proyecto y alumno es de muchos a uno, por tanto en el modelo proyecto.php, se debe incluir esta condición, es decir tendremos que validar la relación entre tablas para que sea muchos a uno. Véase la figura 15.

```
<?php
class Proyecto extends AppModel{
   var $name = 'Proyecto';
   var $belongsTo = array('Alumno' =>
                              array('className' =>'Alumno',
                                    'conditions'=>'',
                                    'order'     =>'',
                                    'foreignKey'=>'alumno_id'
                              )
                     );
}
?>
```

Figura 15. Validar relación muchos a uno

'className' (required): nombre del modelo con el que se quiere asociar. En este caso con 'Alumno'.

'conditions': condiciones SQL que definen la relación.

'order': se utilizaría para ordenar la asociación en el modelo.

'foreignKey': es el nombre de la FK que apunta al modelo con el que está asociado. En este caso en la tabla proyecto debe haber un campo con el nombre alumno_id, pues éste será la FK que habrá que escribir en "foreign key".

```
Array(
   [Proyecto] => Array(
                     [id] => 4
                     [titulo] => Aplicaciones Web
                     [num_paginas] => 200
```

```
                    [alumno_id] = 25
                )
    [Alumno] => Array(
                    [id] => 25
                    [nombre] => Carlos
                    [apellido] => Lirón López
                    [fecha_nacimiento] => 1982-11-11
                    [email] => carlosliron@alu.umh.es
                    [telefono] => 965350895
                    [año_matriculacion] => 2003
                    [nota_selectividad] => 9,8
                )
)
```

Figura 16. Resultado de la relación N:1 en Alumno y Proyecto

- hasAndBelongToMany: N:N, de muchos a muchos. Siguiendo el ejemplo, se dirá que la relación entre proyecto y alumno es de muchos a muchos, por ello lo primero que hay que hacer es crear otra tabla llamada "alumnos_proyectos". Después tendremos que validar la relación de N:N en el modelo alumno.php. Véase figura 17.

'className' (required): nombre del modelo con el que se quiere asociar. En este caso con 'Proyecto'.

'joinTable': anteriormente se ha comentado que en una relación N:N se debe crear otra tabla con el nombre de ambas y las FK de ambas (alumno_id, proyecto_id), también se ha comentado que el nombre de esta tabla debería estar en plural y en orden alfabético. Pues mediante esta opción "joinTable" permite poner el nombre de la tabla aunque no haya seguido el convenio de CakePHP. Por ejemplo si en vez de llamarse alumnos_proyectos se hubiese llamado universitario, mediante esta opción, CakePHP podría trabajar con esa tabla.

```
<?php
class Alumno extends AppModel{
   var $name = 'Alumno';
   var $hasAndBelongsToMany =
           array('Proyecto' =>
                      array('className'=>'Proyecto',
                            'joinTable'            =>'alumnos_proyectos',
                            'foreignKey'           =>'alumno_id',
                            'associationForeignKey'=>'proyecto_id',
                            'conditions'           =>'',
                            'order'                =>'',
```

```
                            'limit'                 =>'',
                            'unique'                =>true,
                            'finderQuery'           =>'',
                            'deleteQuery'           =>''
                        )
            );
}
?>
```

Figura 17. Validar relación muchos a muchos

foreignKey: nombre de la FK en el 'join table' que apunta al modelo actual. En este caso como el modelo es alumno, la FK será alumno_id.

associationForeignKey: nombre de la FK en el 'join table' que apunta al modelo asociado. En este caso como el modelo asociado es proyecto, pues la FK será proyecto_id.

'conditions': condiciones SQL que definen la relación.

'order': se utilizaría para ordenar la asociación en el modelo.

'limit': el número máximo de asociaciones que se quieren en un modelo.

unique: si es *'true',* duplica la asociación entre objetos, con lo cual serán ignorados los métodos *'accessors'* (metodos que acceden al contenido de los objetos pero no los pueden modificar) *y 'query'* (es una consulta, una búsqueda en la base de datos().

'finderQuery': especifica una sentencia completa en SQL para buscar asociaciones.

'deleteQuery': sentencias en SQL que pueden eliminar asociaciones entre modelos de muchos a muchos.

```
Array(
   [Alumno] => Array(
```

```
                    [id] => 25
                    [nombre] => Carlos
                    [apellido] => Lirón López
                    [fecha_nacimiento] => 1982-11-11
                    [email] => carlosliron@alu.umh.es
                    [telefono] => 965350895
                    [año_matriculacion] => 2003
                    [nota_selectividad] => 9,8
                )
    [Proyecto] => Array(
                    [0] => Array(
                                [id] => 4
                                [titulo] => Aplicaciones Web
                                [num_paginas] => 200
                            )
                    [1] => Array(
                                [id] => 20
                                [titulo] => Bases de Datos
                                [num_paginas] => 150
                            )
                )
)
```

Figura 18. Resultado de la relación N:N en Alumno y Proyecto

3. DESARROLLO CON '*SCAFFOLD*'

Scaffold (o '*scaffolding*', del inglés, andamio o andamiaje) es una utilidad de CakePHP que se utiliza para desarrollar aplicaciones Web muy fácilmente. La idea es poder disponer de todos los formularios para inserción, edición y borrado de los registros de cualquier tabla de la base de datos; mediante este método, CakePHP genera automáticamente los listados y formularios necesarios para ello. Los pasos a seguir para ello serían:

1) Crear la tabla '*temas*' (tabla de ejemplo) en la base de datos, como muestra en la figura 19. Los nombres de las tablas deben estar siempre en plural.
2) Crear el controlador, donde se incorpora la opción scaffolding, como se muestra en la figura 20. Todos los controladores se crearán en la carpeta '*[…]/proyecto_cake/app/controllers/*'. El nombre del fichero del controlador para la tabla 'temas' deberá ser '*temasController*'.

```
CREATE TABLE `temas`(
   `id` int(10) unsigned NOT NULL auto_increment,
   `descripcion` text(100) NOT NULL,
   `tema_id` int(10),
```

```
   PRIMARY KEY (`id`)
);
```

Figura 19. Tabla de 'tema'

```
<?php
class temasController extends AppController {
   var $name = 'temas';
   var $scaffold;
}
?>
```

Figura 20. Controlador para la tabla 'temas'

3) Crear el modelo de la aplicación como se indica en la figura 21. Todos los modelos se crearán en la carpeta '*[…]/proyecto_cake/app/models/*'. Ojo, la clase para el modelo de la tabla 'temas' deberá tener el mismo nombre pero en singular, y la propiedad '*$name*' debe tener el valor 'Tema' (en singular y la primera letra mayúscula). El nombre del fichero para el modelo de la tabla 'temas' deberá ser '*tema.php*'.

```
<?php
class tema extends AppModel{
   var $name = 'Tema';
}
?>
```

Figura 21. Modelo de la tabla 'temas'

Tras los pasos anteriores, escribiendo '*http://localhost/proyecto_cake/temas*' dispondremos de las opciones para gestionar la tabla 'temas' de nuestra base de datos (figura 22).

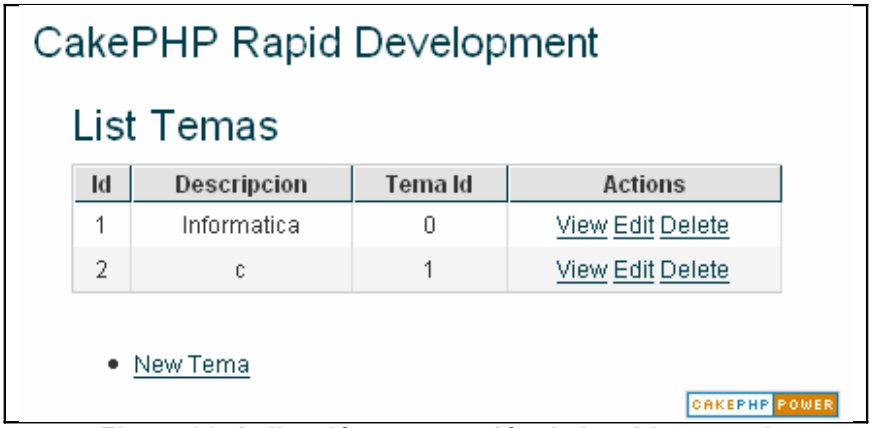


Figura 22. Aplicación para gestión de la tabla 'temas'

4. EJEMPLO COMPLETO

Ahora se va a explicar la forma en la que se puede realizar una pequeña aplicación Web. Pero como antes ya se ha mencionado, esta herramienta trata cuatro tipos de relaciones, por esta razón se va a dividir la base de datos en fragmentos, para ir explicando cada relación por separado.

Relación de muchos a muchos: por ejemplo, libros y autores

Primer paso

Crear la base de datos, para CakePHP no es necesario que se haga con InnoDb, tampoco se tienen que poner los campos relacionados en ambas tablas, es decir, en la tabla autor no hay que poner libro_id, ni en libro hay que poner autor_id, solamente hay que construir otra tabla donde estén estas dos foreign key.

En la figura 23 hay una muestra de dos tablas relacionadas de muchos a muchos. A continuación se va a ir explicando como se haría para la tabla autor, pero también se haría exactamente igual para libro, solamente habría que cambiar los nombre de las tablas.

Segundo paso

Crear los controladores de ambas tablas, recordar que el nombre de los controladores deben ponerse en plural (pero el plural hay que formarlo en inglés), por tanto dentro de la carpeta '*[…]/proyecto_cake/app/controllers/*', hay que crear dos archivos: **autors_controller.php** y **libros_controller.php.** Escribiendo en cada una de ellas el código que se realizará como el que se muestra en la tabla figura 24.

```
CREATE TABLE `autors`(
   `id` int(11) NOT NULL auto_increment,
   `nombre` varchar(50) NOT NULL,
   `apellido` varchar(50) NOT NULL,
   `fecha_nacimiento` date NOT NULL,
   `fecha_defuncion` date default NULL,
   PRIMARY KEY (`id`)
);
```

```
CREATE TABLE `libros`(
   `id` int(10) unsigned NOT NULL auto_increment,
   `titulo` varchar(45) NOT NULL,
   `num_paginas` varchar(45) NOT NULL,
   `editorial_id` int(8) unsigned NOT NULL,
   PRIMARY KEY (`id`),
);

CREATE TABLE `autors_libros`(
   `id` int(11) NOT NULL auto_increment,
   `autor_id` int(11) NOT NULL,
   `libro_id` int(11) NOT NULL,
   PRIMARY KEY  (`id`)
);
```

Figura 23. Relación de tablas de N a N

```
<?php
class AutorsController extends AppController{
   var $name = 'Autors';
}
?>
```

Figura 24. Controlador de Autor

Por otra parte, teniendo en cuenta que posteriormente en el formulario se querrán incorporar funciones que están en “htmlhelper” (ayuda para crear formularios), como son las funciones password, submit, input, etc. Por ello se incorporará en el controlador una función, quedando finalmente como se muestra en la figura 25.

```
<?php
class AutorsController extends AppController{
   var $name = 'Autors';
   var $helpers = array('Html', 'Form');
}
?>
```

Figura 25. Controlador de Autor

Tercer paso

Crear los modelos de ambas tablas, estos archivos se llamarán con el nombre de la tabla en singular seguido de *‘php’*, siguiendo el ejemplo, serán; *‘autor.php’*,

'libro.php'. Estos archivo se crearan dentro de la carpeta *'[…] /proyecto_cake/app/models/'*.

En la figura 26, se muestra como se hace un modelo de una tabla, pero un modelo muy básico, sobre él, posteriormente se irán insertando funciones, así como para validando campos o relaciones entre tablas.

```
<?php
class Autor extends AppModel{
   var $name = 'Autor';
}
?>
```

Figura 26. Modelo de Autor

Debido a que autor es una tabla relacionada N:N con libro, esto tendríamos que identificarlo en el modelo. Esta relación se llama hasAndBelongToMany, en la figura 24 se puede ver un ejemplo con la tabla autor.

```
<?php
class Autor extends AppModel{
   var $name = 'Autor';
   var $hasAndBelongsToMany =
        array('Libro' =>
               array('className' => 'Libro',
                     'joinTable' => 'autors_libros',
                     'foreignKey' => 'autor_id',
                     'associationForeignKey' => 'libro_id'
               )
        );
}
?>
```

Figura 27. Modelo de Autor

En la figura 27, se añade la relación de muchos a muchos, donde se dice que está relacionada con la tabla libro, mediante la tabla *'autors_libros'*, también se escribe la foreign key (autor_id) y la foreign key del modelo asociado (libro_id). Pero si se deja tal como esta ahora, en la tabla libro se verá el identificador del autor y no el nombre, que es lo que verdaderamente interesa, por ello se agregará una función más al modelo, obligándole así que en la asociación de estas dos tablas se pueda ver el nombre y no un número o identificador. Esto mismo también se hará en la tabla libro, donde en vez de querer la id se quiere que el campo que se muestra sea el titulo del libro. Por tanto se tendrá que agregar la

función '*var $displayField= 'campo_tabla";* a ambos modelos, poniendo en cada modelo el campo que se quiere ver en la tabla relacionada.

```
//En el modelo Autor
var $displayField= 'nombre';

//En el modelo Libro
var $displayField= 'titulo';
```

Figura 28. Campo a mostrar en las tablas relacionadas

Cuarto paso

Crear la página principal de ambas tablas. Para ello se creará una función en el controlador del autor, '*[…]/proyecto_cake/app/controllerautor_controller.php/*'., escribiendo el código que se muestra en la figura 29.

```
function index(){
   $this->Autor->recursive = 0;
   $this->set('autors', $this->Autor-> findAll());
}
```

Figura 29. Función index de autor

Una vez creada la clase, se creará el diseño de la página principal de Autor. Para ello se tiene que crear una carpeta llamada autors, dentro '*[…] /proyecto_cake/app/view/*', quedando, '*[…]/proyecto_cake/app/view/* app/view/autors, creando dentro de la carpeta otro archivo llamado index.thtml, donde se indican: el diseño de la página principal, los campos de las tablas, así como los enlaces. Se adjunta un ejemplo en las figuras 30 y 31.

```
<h1>Lista de Autores</h1>
<table>
//Titulo de la tabla, con los campos de la misma.
<tr>
    <th>Id</th>
    <th>Nombre</th>
    <th>Apellido</th>
    <th>Fecha de Nacimiento</th>
    <th>Fecha de Defunción</th>
    <th>Cambios</th>
</tr>
//Datos de la base de datos por filas
<?php foreach($autors as $row)?>
<tr>
    <td><?php echo $row['Autor']['id'];?></td>
    <td><?php echo $row['Autor']['nombre'];?></td>
    <td><?php echo $row['Autor']['apellido'];?></td>
```

```
    <td><?php echo $row['Autor']['fecha_nacimiento'];?></td>
    <td><?php echo $row['Autor']['fecha_defuncion'];?></td>
//Enlaces para: editar, eliminarlo o ver un registro.
    <td><?php echo $html->link('Vista',
                               '/autors/view/'.$row['Autor']['id']);?>
        <?php echo $html->link('Editar',
                               '/autors/edit/'.$row['Autor']['id']);?>
        <?php echo $html->link('Eliminar',
                               '/autors/delete/'.$row['Autor']['id']);?>
    </td>
</tr>
<?php endforeach;?>
</table>
//Link para crear un dato Nuevo, en este caso un autor
<ul>
    <li><?php echo $html->link('Nuevo Autor','/autors/add');?></li>
</ul>
```

Figura 30. Diseño de la página autor

Como se puede ver en la figura 31, hay cuatro enlaces, vista, editar, eliminar y nuevo, a continuación iré se crearán las funciones para que funcionen.

Lista de Autores

Id	Nombre	Apellido	Fecha de Nacimiento	Fecha de Defunción	Cambios
1	Marta	Martínez García	1948-12-01	0000-00-00	Vista Editar Eliminar
2	Inma	López Pamies	1954-11-04	1985-10-08	Vista Editar Eliminar
33	kkk	kklkl	0000-00-00	0000-00-00	Vista Editar Eliminar
32	wdwed	dcsdcsd	1970-01-01	1970-01-01	Vista Editar Eliminar
5	Luis	Pérez Peiro	1971-07-07	0000-00-00	Vista Editar Eliminar
6	Jose	Oltra Segura	1950-06-08	0000-00-00	Vista Editar Eliminar
7	Paco	Segura Sebastian	1954-05-10	0000-00-00	Vista Editar Eliminar
8	Antonio	Poveda Ricarte	1953-04-11	1980-09-28	Vista Editar Eliminar
9	Carlos	Gómez Ortín	1950-03-12	2004-10-01	Vista Editar Eliminar
10	Juan	García Herrero	1970-01-13	2006-10-26	Vista Editar Eliminar

- Nuevo Autor

Figura 31.Página Web de autor

Quinto paso

Ahora se creará la vista de la tabla autor. Primero, al igual que antes se escribirá la función view (figura 32), dentro del controlador de la tabla autor (‘*[…] /proyecto_cake/app/controlles*’), y a continuación el diseño de la pagina que se

creará dentro de '*[…]/proyecto_cake/app/views/autors*' y se llamará view.thtml (figura 33).

```
Function view($id){
   $this->set('autors', $this->Autor->read(null, $id));
}
```

Figura 32. Función view en el controlador

```
<h1>Vista del Autor</h1>
<table>
//Haremos lo siguiente para todos los campos de la tabla
<tr height="40px" width="100px">
    <td><?php echo 'Id:';?></td>
    <td><?php echo $autors['Autor']['id'];?></td></tr>
</table>
//Crear los enlaces necesarios
<ul>
    <li><?php echo $html->link('EditarAutor',
                               '/autors/edit/'.$autors['Autor']['id']) ?></li>
</ul>
...
```

Figura 33. Diseño de la vista del autor

Vista del Autor

Id:	1
Nombre:	Marta
Apellido:	Martínez García
Fecha de Nacimiento:	1948-12-01
Fecha de Defunción:	0000-00-00

Figura 34.Página Web de la vista del autor

Sexto paso

A continuación se creará la función para crear un autor nuevo (función add). Como anteriormente, primero se crea la función en el controlador y posteriormente el diseño. Ver figuras 35, 36 y 37.

En la figura 35 se muestra una función para crear un nuevo autor, cuando la tabla no tiene ninguna relación con otra tabla. Pero la tabla "autor" esta relacionada con otra tabla, entonces hay que crear una variable autorArray para que se pueda ver en una tabla los datos de la otra, como en la figura 36, que sería la función completa, donde se indica en negrita el código necesario para una relación N:N.

```
function add()
{
   if(empty($this->data)){
      $this->set('autors', null);
   }
   else
   {
      $this->cleanUpFields();
      if($this->Autor->save($this->data))
      {
         //redirecciona a otra página diciendo que se ha guardado
         //$this->flash('Usuario guardado.','/autors/index');

         //redirecciona a la misma página
         $this->redirect('autors/index');
      }
      else
      {
         $data = $this->data;
         $this->set('autors', $data);
      }
   }
}
```

Figura 35.Función para añadir un autor. Sin tablas relacionadas

```
function add()
{
   if(empty($this->data))
   {
      $this->set('libroArray', $this->Autor->Libro->generateList(''));
      $this->set('selectedLibro', null);
   }
   else
   {
      $this->cleanUpFields();
      if($this->Autor->save($this->data))
      {
        $this->redirect('autors/index');
      }
      else
      {
         $data = $this->data;
         $this->set('autors', $data);
         $libro = null;
         $this->set('libroArray',$this->Autor->Libro->generateList());
         if(isset($data['Libro']['Libro']))
         {
            foreach($data['Libro']['Libro'] as $var)
            {
               $libro[$var] = $var;
            }
         }
         $this->set('selectedLibro', $libro);
      }
   }
}
```

Figura 36. Función para añadir un autor. Con tablas relacionadas

Una vez realizada la función, hay que hacer el diseño, al igual que antes, se creará el archivo add.thml dentro de '*[…]/proyecto_cake/app/view/autors*'.

En la figura 37, se ha escrito el código para hacer el diseño. A continuación se explicará algunas funciones de la figura anterior, como son las siguientes:

```
<div class="required">
```

En la línea anterior se especifica si el campo es opcional u obligatorio (requerido). Si es opcional se pondrá optional, y si es obligatorio se escribirá required. Pero con poner "required" en el código no es suficiente para obligar al usuario a escribir algo en dicho campo, para ello tendremos que volver al modelo de la tabla y validar los campos obligatorios, como se muestra en la figura 38.

```
<h1>Nuevo Autor</h1>
<form action="/proyecto_cak/autors/add" method="POST">

/******************Crear campo para el nombre*****************/
<div class="required">
    <label for="autor_nombre">Nombre:</label>
    <?php echo $html->input('Autor/nombre', array('size' => '50'));?>
    <?php echo $html->tagErrorMsg('Autor/nombre','Nombre no válido');?>
<br>
</div>
/************************************************************/

/******************Crear campo de fecha***********************/
<div class="optional">
    <label for="autor_fecha_defuncion">Fecha de Defunción:</label>
    <?php echo $html->dateTimeOptionTag('Autor/fecha_nacimiento',
                                        'DMY','NONE');?>
</div>
/************************************************************/

/**********Crear campo relacionado con otra tabla*************/
<div class="optional">
    <label for="libro_libro">Libros relacionados:</label>
    <?php echo $html->selectTag('Libro/Libro', $libroArray,
                                $selectedLibro,
                                array('multiple'=>'multiple',true,
                                'size' => '10'), null, false);?>
    <?php echo $html->tagErrorMsg('Libro/Libro','Fecha: YYYY-MM-DD');?>
<br>
</div>
/************************************************************/
<div class="submit"><input type="submit" value="Add" /></div>
</form>
<ul>
<li><?php echo $html->link('Lista de autores', '/autors/index')?></li>
</ul>
```

Figura 37. Diseño para crear un nuevo autor

```
var $validate = array('nombre'=>VALID_NOT_EMPTY,
                      'apellido'=>VALID_NOT_EMPTY,
                      'fecha_nacimiento'=>VALID_NOT_EMPTY,);
```

Figura 38. Validar campos del formulario autor

Una vez explicado como se validan los campos, se continuará en la figura 37. Como se ha dicho, la primera línea se utiliza para decir si un campo es obligatorio u opcional, a continuación la segunda línea sirve para poner el nombre del campo en la tabla, como por ejemplo la línea siguiente.

```
Label for="autor_nombre">Nombre:</label>
```

En la línea siguiente se especifica el tipo de campo necesario, en este caso para escribir el nombre el autor, se necesita un cuadro de texto, que se llama input, como en la siguiente línea de código.

```
<?php echo $html->input('Autor/nombre',array('size'=>'50'));?>
```

Si en vez de ser un campo de texto fuese una fecha, el campo se llamaría “dateTimeOptionTag”, como se muestra en el campo “fecha de nacimiento” de la figura 36, donde también se puede especificar el orden de la fecha (date) “DMY”, “YMD”, “MDY”, y el tiempo (time) se puede poner: “12” (12 horas), “24” (24 horas) o “none” (para que no aparezca el tiempo).

En el caso que fuese un campo de otra tabla, como en el caso libro-autor, el campo necesario es selectTag, seguido de la tabla donde esta / y otra vez la tabla donde esta situado, añadiendo $nombre_de_la_tabla_array, (variable definida en el controlador), seguido de $selected_nombre_de_la_tabla, como se muestra en la siguiente línea.

```
<?php
echo $html->selectTag('Libro/Libro', $libroArray, $selectedLibro,
                      array('multiple'=>'multiple',true,'size'=>'10'),
                      null, false);
?>
```

Por ultimo solo faltaría escribir un mensaje de error para indicarle al navegador que los datos del formulario los ha escrito incorrectamente. Esto se haría con una línea de código, como se muestra a continuación.

```
<?php echo $html->tagErrorMsg('Autor/nombre', 'Nombre no válido');?>
```

Séptimo paso

Ahora se creará la función *'edit'*, para modificar los datos, esta función es muy parecida a la anterior con la peculiaridad que en ésta, se tiene que pasar la *'id'* del formulario, para poder ver los datos y modificarlos. En la figura 40, se muestra la modificación respecto a la función add, a partir del "else" se haría lo mismo para ambas funciones, *'add'* y *'edit'*.

Para hacer el diseño de la pagina *'edit'* (modificar), se debe hacer el mismo formulario que para *'add'* (nuevo), con la diferencia que se tiene que pasar la *'id'* en la primera línea de código, tal como se muestra en la figura 41 y a continuación sería igual que en el código de *'edit'*. Este archivo se debe crear como el resto, dentro de '*[…]/proyecto_cake/app/view/autors*' y llamarse *'edit.thtml'*.

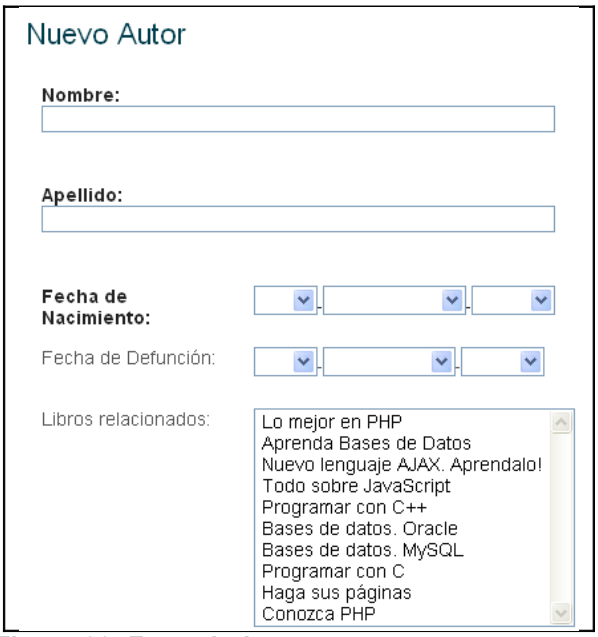


Figura 39. Formulario para crear un nuevo autor

```
function edit($id = null){
   if(empty($this->data)){
      $this->Autor->id = $id;
      $this->data = $this->Autor->read();
      $data= $this->Autor->read(null, $id);
      $this->set('libros', $data);
      $libro = null;
      $this->set('libroArray', $this->Autor->Libro->generateList());
      foreach($data['Libro'] as $var){
         $libro[$var['id']] = $var['id']; }
```

```
          $this->set('selectedLibro', $libro);}
   else{
//a partir de aquí igual que la función add)
```

Figura 40. Función editar para autor

```
<?php echo $html->formTag('/autors/edit/'.$html-
>tagValue('Autor/id'));?>
```

Figura 41. Código para leer una id

Figura 42. Formulario para modificar un autor

Octavo paso

Por último quedaría por hacer la función *'delete'* (eliminar), la cual debe introducirse también dentro del controlador autor y la función será la que adjunto en la figura 43.

```
Function delete($id){
  $this->Autor->del($id);
  $this->redirect('autors/index');}
```

Figura 43. Función para borrar los autores

Relación de muchos a uno: aquí se estudiará un ejemplo de relación N:1, para ello se crearan dos tablas: editorial y libro (una editorial, muchos libros). Al ser una relación de N:1, los formularios de la tabla editorial no tendría ninguna peculiaridad, mientras que los de libro si, la página de editoriales se haría normal mientras que la de libro sería un poco diferente. Ya que en libro habría que crear

la relación entre tablas, indicando el tipo de relación que es, y posteriormente en la tabla libro estaría la clave foránea a editorial, la cual tendríamos que indicarla.

Primer paso

Crear la tabla libro, como en ella, editorial es una foreign key, pues dicho campo habría que escribirlo en singular y seguido de *'id'*, de la siguiente forma: *'editorial_id'*. En la figura 44 se muestra la tabla libro

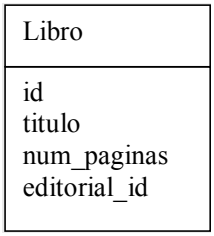


Figura 44. Tabla libro

Segundo paso

Crear el controlador de libro, se haría igual que en el ejemplo anterior

Tercer paso

Crear el modelo del libro, teniendo en cuenta que es una relación N:1, por lo tanto, se creará la relacion BelongsTo en el modelo. Ver figura 45.

```
var $belongsTo = array(
            'editorial' =>
             array('className' => 'Editorial',
                   'conditions' => '',
                   'order' => '',
                   'foreignKey' => '',
                   'counterCache' => ''),);
```

Figura 45. Relación N:1 en Libro

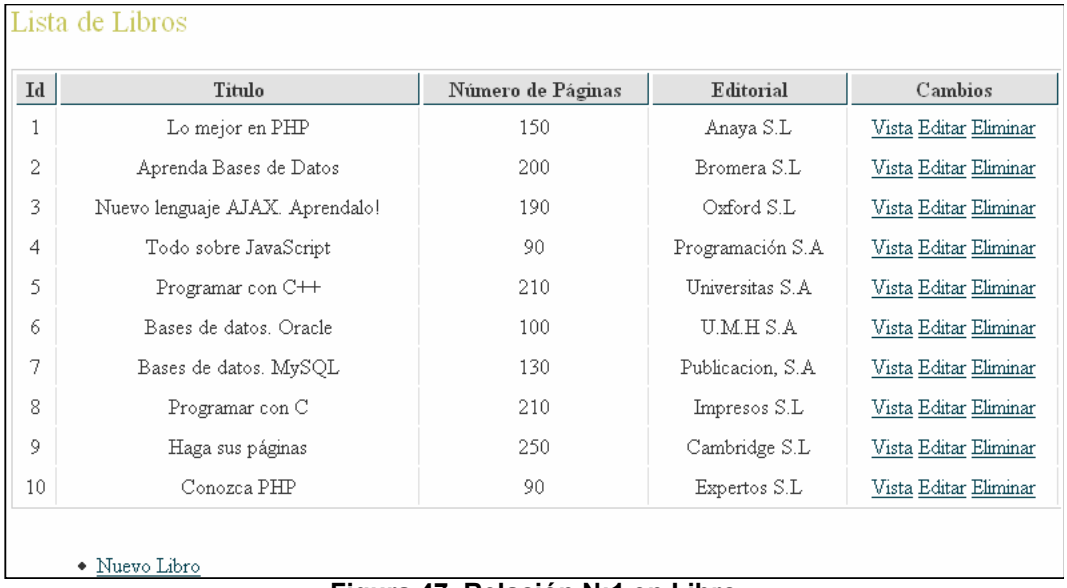
Cuarto paso

Crear la página principal, para ello se creará el index en el controlador de libro, y luego la pagina index.thtml(como en el ejemplo de N:N). Pero en el index thtml se tendrá en cuenta que el campo editorial es una FK, por lo tanto no se puede poner $row[nombre de la tabla][nombre del campo] de esta tabla, sino el nombre de la tabla editorial y el nombre del campo de la tabla editorial también, quedando como en la figura 46.

```
<td><?php echo $row['Libro']['id'];?></td>
<td><?php echo $row['Libro']['titulo'];?></td>
<td><?php echo $row['Libro']['num_paginas'];?></td>
<td><?php echo $row['editorial']['nombre'];?></td>
```

Figura 46. Mostrar campo de otra tabla

Dando como resultado un formulario como el que se muestra en la figura 44.



Lista de Libros

Id	Titulo	Número de Páginas	Editorial	Cambios
1	Lo mejor en PHP	150	Anaya S.L	Vista Editar Eliminar
2	Aprenda Bases de Datos	200	Bromera S.L	Vista Editar Eliminar
3	Nuevo lenguaje AJAX. Aprendalo!	190	Oxford S.L	Vista Editar Eliminar
4	Todo sobre JavaScript	90	Programación S.A	Vista Editar Eliminar
5	Programar con C++	210	Universitas S.A	Vista Editar Eliminar
6	Bases de datos. Oracle	100	U.M.H S.A	Vista Editar Eliminar
7	Bases de datos. MySQL	130	Publicacion, S.A	Vista Editar Eliminar
8	Programar con C	210	Impresos S.L	Vista Editar Eliminar
9	Haga sus páginas	250	Cambridge S.L	Vista Editar Eliminar
10	Conozca PHP	90	Expertos S.L	Vista Editar Eliminar

• Nuevo Libro

Figura 47. Relación N:1 en Libro

En la figura 47, y en concreto en la columna de editorial, se puede ver que en editorial no muestra los “identificadores” (1, 2, 3,…) sino los nombres de las editoriales.

Quinto paso

En este paso se creará la vista del libro, se hará exactamente igual que en el ejemplo anterior, pero en el *‘view.thtml’* (el diseño de libro), se modificará el campo

de la editorial ya que de lo contrario mostraría el identificador. Además de agregar eso también se puede agregar un link en html para que al pulsar redireccione a la página de la vista de la editorial. Después de los cambios, el formulario se quedaría como en la figura 48.

```
<tr><td><?php echo 'Editorial:' ?></td>
<td><?php echo $html->link($libros['editorial']['nombre'],
"/editorials/view/".$libros['Libro']['editorial_id']);?></td>
</tr>
```

Figura 48. Mostrar el nombre de las editoriales en libro

Una vez realizado quedará como en la figura 49

Sexto paso

En este paso se tendrá que crear la función dentro del controlador para añadir libros nuevos, esta función cambiará, con respecto a las anteriores. Ver figura 50.

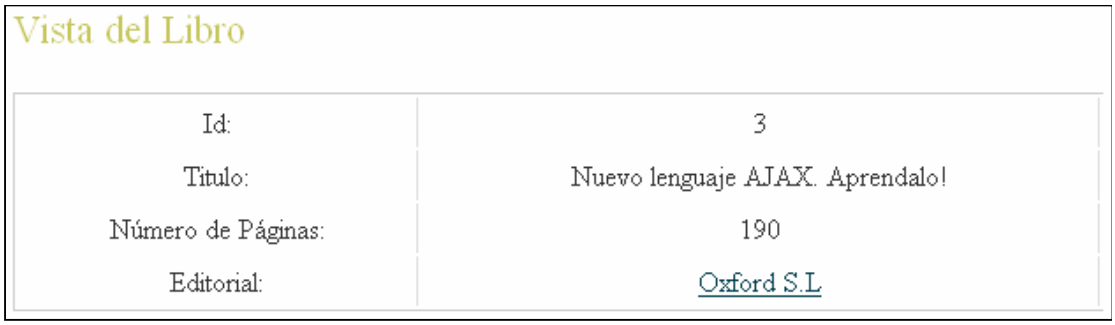


Figura 49. Vista del Libro

```
function add (){
   if(empty($this->data)){
      $this->set('editorialArray',
                  $this->Libro->Editorial->generateList());
      $this->set('libros', null);
   }
   else{
      if($this->Libro->save($this->data))
         $this->redirect('libros/index');
      else{
         $data = $this->data;
         $this->set('libros', $data);
         $libro = null;
         $this->set('editorialArray',
                     $this->Libro->Editorial->generateList());
      }
   }
}
```

Figura 50. Función 'nuevo' (add) del libro

Respecto a la página add.thtml se quedaría como en el ejemplo anterior, haciendo un pequeño cambio en el campo editorial. En dicho campo se necesitará un select, por ello se escribirá *'selectTag'* seguido del nombre_tabla/nombre_campo de la tabla relacionada. Ver figura 51.

```
<div class="optional">
      <label for="libro_editorial_id">Editorial:</label>
      <?php echo $html->selectTag('Libro/editorial_id',
      $editorialArray,
      array('id' => 'libro_editorial_id', 'size' => '1')); ?>
</div>
```

Figura 51. Escribir una FK en un formulario

Séptimo paso.

Al igual que en el sexto paso habría que cambiar las funciones en la que aparece editorial, por tanto se cambiará tanto el controlador como la función edit.thtml. Ver figuras 52 y 53.

```
Function edit($id)
{
   if(empty($this->data))
   {
      $this->Libro->id = $id;
      $this->data = $this->Libro->read();
      $data =$this->Libro->read(null, $id);
      $this->set('libros', $data);
      $this->set('editorialArray',
                 $this->Libro->Editorial->generateList());
   }
   else
   {
      if($this->Libro->save($this->data))
      {
         $this->redirect('libros/index');
      }
      else
      {
         $data = $this->data;
         $this->set('libros', $data);
         $this->set('editorialArray',
                    $this->Libro->Editorial ->generateList());
      }
   }
}
```

Figura 52. Función editar de Libro

```
<div class="optional">
   <label for="libro_editorial_id">Editorial</label>
<?php
echo $html->selectTag('Libro/editorial_id', $editorialArray,
                      $libros['Libro']['editorial_id'],
                      array('id'=>'libro_editorial_id','size'=>'0'));
?>
</div>
```

Figura 53. Diseño de la página para editar el libro

En la figura 53, además de poner el *'selectag'*, también tendríamos que pasarle la id de ese libro y además debería mostrarlo, por ello la función no es igual en nuevo y en editar.

Tabla Usuario

Para realizar la tabla usuarios se hará igual que las anteriores, por ello solo se comentarán las peculiaridades de esta tabla.

El campo mas importante de esta tabla es el de la contraseña. Para que en dicho campo "password" escriba asteriscos en vez de letras es suficiente con indicarlo en el formulario, poniendo password delante del campo, como se indica a continuación.

```
<?php
echo $html->passwordTag('Usuario/password',array('size'=>'40'));
?>
```

Figura 54. Campo password en el formulario de editar y nuevo(edit y add)

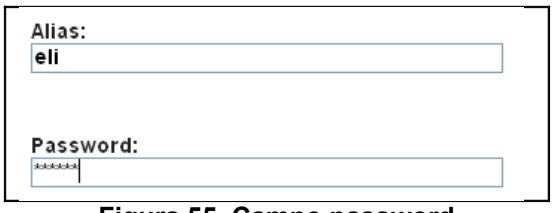


Figura 55. Campo password

También se puede encriptar la contraseña, para que en la base de datos no se pueda ver, mediante la función en PHP que se adjunta en la figura 56.

```
$this->data['Usuario']['password'] = md5($this->
data['Usuario']['password']);
If($this->Usuario->save($this->data['Usuario']))
```

Figura 56. Función para encriptar

La función de la figura 56, se escribir tanto en insertar un campo nuevo, como en editar un campo, quedando en la lista de usuarios, tal como se muestra en la figura 57.

Id	Alias	Password	Nombre	Dirección	Teléfono	E-mail	Cambios
35	admin	6a973a1cb0703afd0a7e65e4c14ea17a	administracción	Bigastro	892739749	administraccion@cake.es	Vista Editar Eliminar

Figura 57. Lista de usuarios

En dicha tabla, entre otros campos, está el campo e-mail, dicho campo se ha validado (en el modelo de la clase usuario) como un campo email, con lo cual se asegura que el campo tenga una "@" y un".", en el caso de faltarle alguna de las dos exigencias de un campo email, mostraría un error como en la figura 58

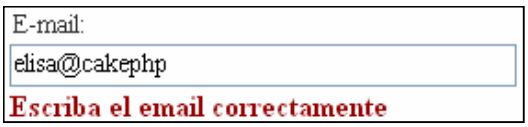


Figura 58. Campo email

5. AÑADIR FUNCIONALIDAD

Debido a la gran flexibilidad que muestra este programa, se va a crear una función más. Dicha función consiste en crear en las vistas de las tablas que están relacionadas mediante una asociación de N:N, todos los campos de una y otra tabla. Por ejemplo dentro de la tabla libro (siguiendo el ejemplo anterior), se mostrará todos los autores, con todos los campos de cada autor. Para ello es suficiente con hacer una función dentro de la vista de libro como la que se muestra en figura 59.

```
<h2>Autores relacionados</h2>
<?php if(isset($libros['Autor']['0'])&&is_array($libros['Autor'])):?>
<table><tr>
<?php foreach ($libros['Autor']['0'] as $column => $value):?>
<th><?php echo $column;?></th>
<?php endforeach;?>
```

```
<th>Acciones</th></tr>
<?php foreach ($libros['Autor'] as $row):?> <tr>
    <?php foreach ($row as $column => $value):?>
        <td><?php echo $value?></td>
    <?php endforeach;?>
<?php if (isset($this->controller->Libro->Autor)):?><td>
    <?php echo $html->link('View','/autors/view/'.
            $row[$this->controller->Libro->Autor->primaryKey]);?></td>
<?php else:?>
<td
    <?php echo $html->link('View','/autors/view/'.
            $row[$this->controller->Libro->primaryKey]);?>
<ul>
<li><?php echo $html->link('Nuevo Autor','/autors/add/');?></li>
</ul>
```

Figura 59. Relación de todos los campos de una tabla

En la figura 60 se muestran los autores relacionados con el libro “lo mejor en PHP”.

Figura 60. Relación entre tablas

6. DISEÑO DE LA APLICACIÓN

A pesar que cakePHP incorpora estilos CSS, se diseñará otro posible estilo de diseño, para ello utilizaremos programas de diseño, estilos CSS, así como tablas, viñetas, etc.

CakePHP lleva incorporado una hoja de estilos situada dentro de ‘*[…] /proyecto_cake/*app/webroot/css’, es una hoja de estilos muy completa, con varios estilos para los campos: input, select, etc

También se va a comentar la forma de hacer los link, esto se haría con la función link, (función ya creada por cakePHP), seguido del nombre que queramos que aparezca en el diseño y después del nombre de la carpeta, es decir el nombre de la tabla, seguido del archivo que queramos enlazar (index, edit, add, view). Como muestro en la siguiente línea.

```
<?php echo $html->link('Autor',"/autors/")?>
```

Para empezar con la aplicación lo primero que se va a diseñar ha sido una cabecera, así como un menú y un pie de página. Todo ello solo se haría en una página, llamada default, que se encuentra en '*[…] /proyecto_cake/*libs/view/templates/layouts', en la figura 61 se adjunta un posible ejemplo de página default.

```
<div id="header">
<?php echo $html->image('titulo.png');?>
</div>
<div>
    <td>
        <ul class="menu">
            <li><?php echo $html->link('Autor',"/autors/");?></li>
        </ul>
    </td>
</div>
<div id="content">
    <?php if ($session->check('Message.flash'))
             $session->flash();
      echo $content_for_layout;?>
</div>
<div id="footer">
<?php echo $html->image('footer.png');?>
</div>
```

Figura 61. Página default

Teniendo realizado el código de la figura 61, en las siguientes páginas solo se creará el contenido de la página, así como por ejemplo crear el contenido de la página principal, solo habrá que ir a la página "home", situada en '*[…] /proyecto_cake/*libs/view/templates/pages y escribir lo necesario, quedando como se puede ver en la figura 62.

Figura 62. Página Principal

En las figuras siguientes se mostrarán las distintas vistas de esta aplicación.

Figura 63. Formulario de una lista de datos

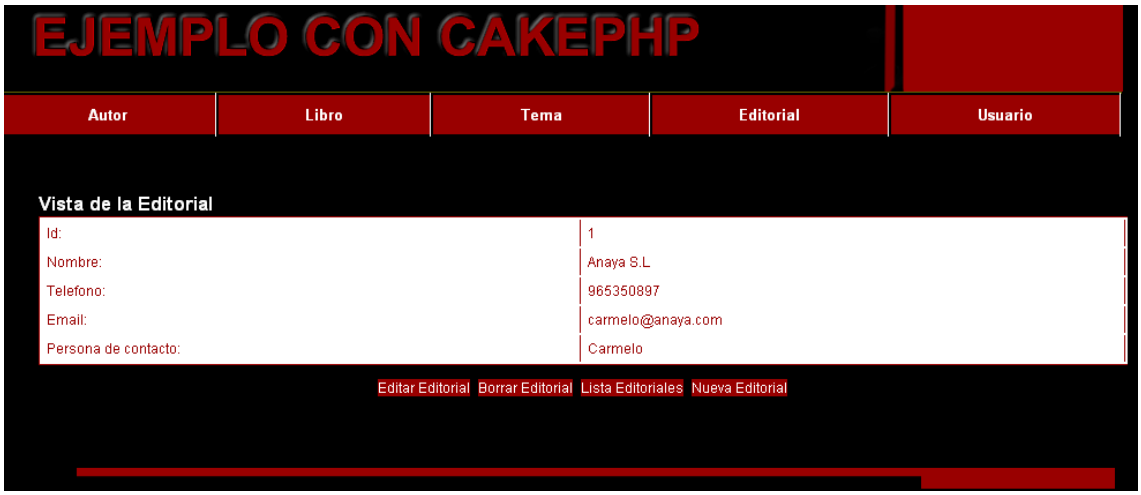


Figura 64. Vista sencilla de un dato de cualquier tabla

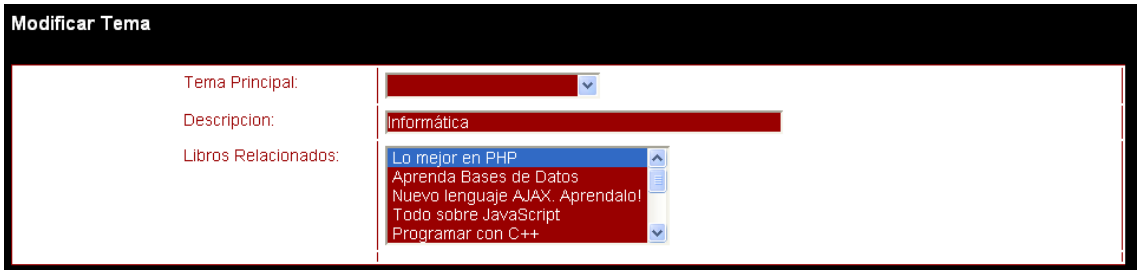


Figura 65. Formulario para editar

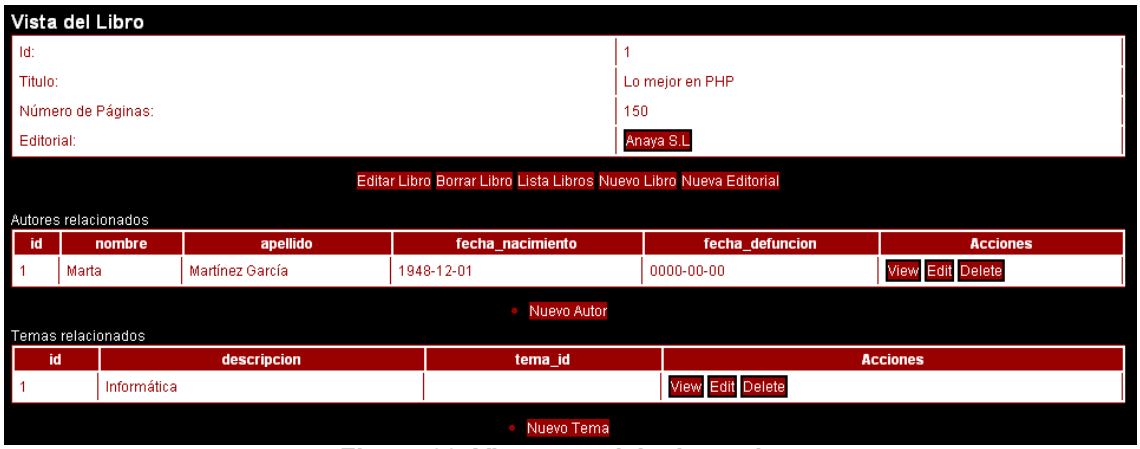


Figura 66. Vista compleja de un dato

En la figura 66, se ven las vistas de un dato relacionado con dos tablas, en este caso la vista de libro donde se ven autores y temas relacionados. Por cada libro seleccionado saldría todos los datos del libro, así como todos los datos del autor relacionado y de todos los temas relacionados también con dicho libro