Práctica completa aplicación Web (II)

H

La segunda aplicación completa es muy didáctica porque concentra muchos de los elementos que componen a las aplicaciones web clásicas:

- Uso de formularios
- Generación dinámica de código HTML
- Uso de clases que representan objetos de base de datos
- Acceso a base de datos
- Mostrar resultados en tablas HTML
- Actualización de datos en tablas de bases de datos
- Directorios de acceso restringido

Aplicación: Noticias comentadas

El objetivo de la aplicación es mantener un sitio web que muestra las noticias de último momento y permite que los lectores puedan incluir comentarios de las noticias. Desde el punto de vista funcional es similar a un weblog o a un foro.

La aplicación contendrá dos partes:

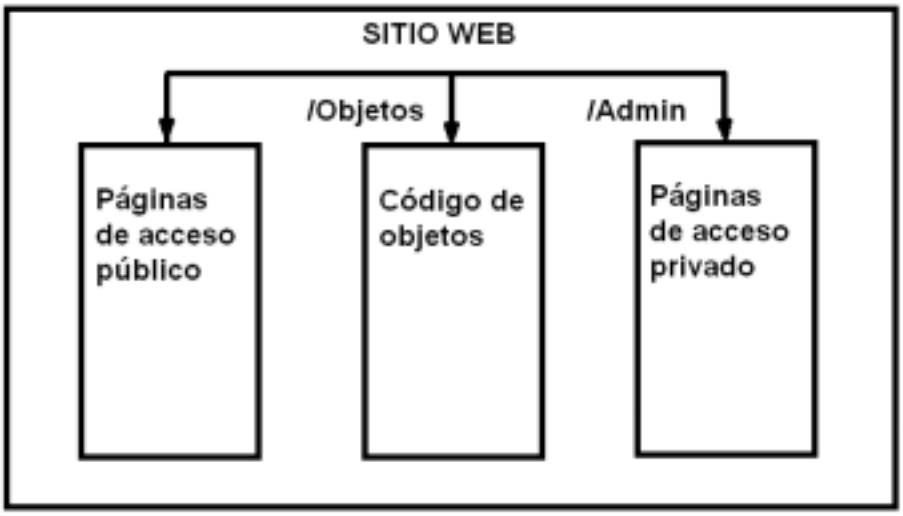
- **Zona del lector:** El lector puede leer las noticias e incluir sus comentarios.
- **Zona del redactor:** El redactor (o administrador) incluye las noticias con posibilidades de corregir o eliminar noticias ya existentes. Esta zona es de acceso restringido por inicio de sesión.

Estructura del proyecto

La aplicación se denominará Noticias y en el servidor se creará un directorio con ese nombre para almacenar todo el código relacionado con la aplicación.

Debajo del directorio Noticias podemos crear una estructura organizativa que nos facilite el mantenimiento del sitio. En el sitio tendremos:

- Archivos php de uso normal; por uso normal se entiende que son páginas de uso irrestricto (carpeta Noticias)
- Archivos php de uso limitado al editor de las noticias (carpeta Noticias\Admin)
- Archivos php de definición de clases (carpeta Noticias\Objetos)



El código de esta aplicación se encuentra en Parte 5/code/Noticias.

Datos del proyecto Noticias comentadas

La base de datos del proyecto se denomina NewsBD y contendrá las dos tablas siguientes:

- Noticia
- Comentario

La información base son todas las noticias publicadas por el editor mientras que los comentarios siempre hacen referencia a una noticia previa y son generados por cualquier lector que quiera aportar su opinión.

Para cada una de las noticias se almacenará un título, un texto, una imagen opcional y la fecha de publicación. Obviamente, será necesario dar un número identificativo único para cada noticia.

Para cada comentario será necesario almacenar un texto, el nombre del lector que hace el comentario, la fecha del comentario y la referencia a la noticia que se está comentando. Como en el caso anterior, será necesario definir un campo clave que identifique de manera única a cada comentario.

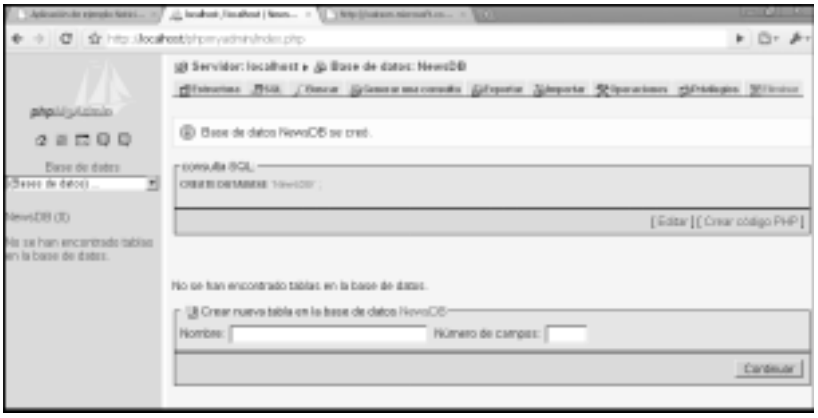
Las tablas Noticia y Comentario están relacionadas por el campo identificador de la noticia, que es clave primaria de la tabla Noticia y clave foránea en la tabla Comentario.

Antes de crear el código para las clases crearemos la base de datos en el gestor de base de datos que hayamos elegido. En este caso lo haremos con MySQL, por lo que emplearemos su gestor PhpMyAdmin.

Nota: PhpMyAdmin es una herramienta escrita en PHP para gestionar la administración de MySQL en la web. Nos permite crear y eliminar bases de datos, crear, eliminar y modificar tablas, y eliminar, modificar e introducir registros de tablas. Además nos permite ejecutar sentencias SQL, administrar permisos y privilegios, exportar datos, etc.

En la siguiente figura se muestra la pantalla inicial del administrador de MySQL, a la cual accedemos con la dirección http://localhost/phpmyadmin/index.php y desde la que podemos iniciar la tarea de creación de la base de datos NewsDB. En este caso se debe observar que para esta base de datos se elige el conjunto de caracteres latin1-spanish-ci que también es la opción elegida para el cotejamiento (ordenamiento).

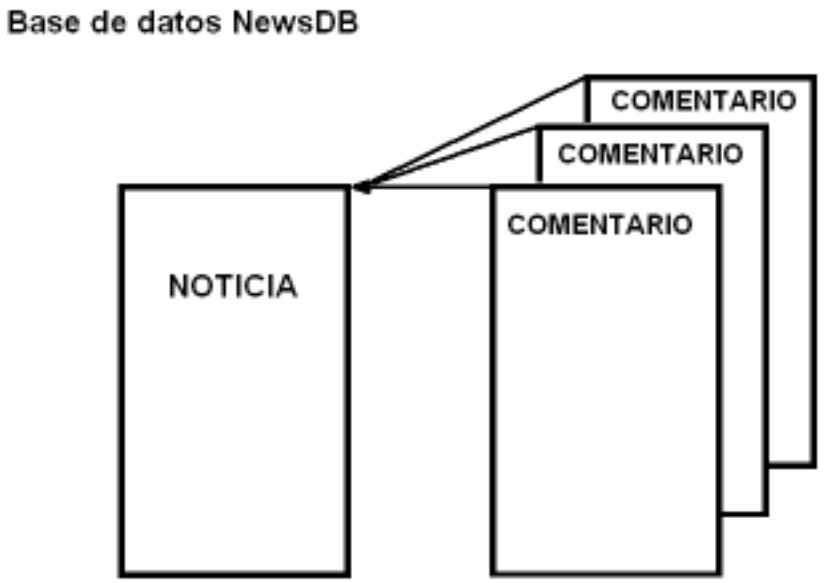
Al crear la base de datos aparece esta pantalla:



Clases del proyecto

El análisis de los datos nos lleva inmediatamente a pensar en las clases. El uso de clases nos permitirá crear un código más fácil de mantener y de comprender, además de dar como resultado programas más compactos.

Al pensar en las funcionalidades del proyecto resulta evidente crear las clases Noticia y Comentario. Pero también necesitaremos listar las noticias como una colección, por lo que también tendremos que crear una clase colNoticias (que es básicamente un conjunto de noticias y no uno sólo como lo es el objeto Noticia).



Cuando consultemos los comentarios de una noticia nos resultará útil contar con una clase que nos entregue la colección de comentarios, por lo que también crearemos una clase colComentario (que es un conjunto de comentarios agrupados por algún criterio: por ejemplo, de la misma fecha, del mismo tema, del mismo autor).

Para gestionar la conexión, desconexión, el conjunto de resultados y la liberación del conjunto de resultados utilizaremos una clase que denominaremos auxiliar de base de datos, AuxDB, y que ya hemos explicado en el capítulo anterior, por lo que aquí simplemente la utilizaremos. Esta clase utiliza la extensión mysqli para acceder al gestor MySQL. Si por alguna razón queremos utilizar otra base de datos u otra extensión tendremos que reemplazar esta clase mientras que el resto de las páginas prácticamente no requerirán cambios.

La definición de nuestra clase auxiliar no comienza con la definición de la propia clase sino con la definición de la interfaz IAuxDB.

La clase auxiliar tendrá estos métodos:

- Conexión con el gestor de base de datos (método conectar)
- Desconexión del gestor de base de datos (método desconectar)
- Ejecución de sentencias SQL (método ejecutarSQL), pudiendo ser sentencias que devuelven filas o no
- Gestión del conjunto de resultados de una consulta

- Obtener la fila siguiente de una consulta (método obtenerFila)
- Conocer la cantidad de filas obtenidas en una consulta (método cantidadFilas)

- Liberar el conjunto de resultados (método liberarRecursos)

En la clase AuxDB debemos modificar los parámetros de conexión, en este ejemplo la base de datos se denomina NewsDB.

Clase Noticia

La clase Noticia nos permitirá crear objetos Noticia y, dada la definición del proyecto Noticias, será necesario que la clase Noticia tenga métodos que nos facilite la lectura, inserción, modificación y eliminación de noticias.

En el capítulo 4 de la parte 4 sobre clases y objetos (página 757) ya hemos visto cómo definir una clase en PHP:

```
<?php
// Noticia.php

class Noticia
{
```

A continuación se definen las variables miembro o propiedades de la clase.

Propiedades de la clase Noticia

Las propiedades de la clase Noticia se corresponden con los campos de la tabla Noticia:

- **AId**: Identificador autoincremento (clave primaria) que nos permite definir de modo único a cada noticia de la tabla. Elegiremos un campo BIGINT, no nulo y de autoincremento.
- **ATitulo**: Es el título de la noticia. Para no restringir la longitud del campo lo definiremos como Text, y no permitiremos que el campo quede con valor nulo (vacío). Puede incluir código HTML.
- **ATexto**: Es el texto de la noticia, por lo que normalmente tendrá un texto bastante extenso. Lo definiremos como Text, y no permitiremos que el campo quede con valor nulo (vacío). Puede incluir código HTML.

- **AFecha**: Es la fecha y hora en la que se creó la noticia. Se informará siempre por lo que nunca admitirá valor nulo.

Siguiendo con la definición de la clase en nuestro código php:

```
private $AId;
private $ATitulo;
private $ATexto;
private $AFecha;


// Las funciones de acceso nos sirven para recuperar los
// valores de las propiedades
function getAId()       {return $this->AId;}
function getATitulo()   {return $this->ATitulo;}
function getATexto()    {return $this->ATexto;}
function getAFecha()    {return $this->AFecha;}
```

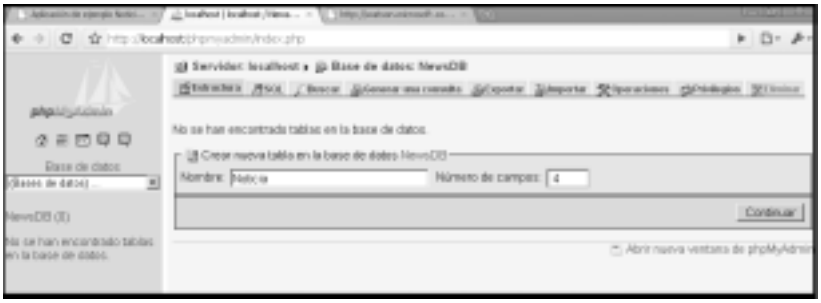
Los objetos se crean mediante el constructor de la clase, la que actúa al crear el objeto, por ejemplo:

```
$obj = New Noticia();  // atención: esta línea no pertenece a la
                       // definición de la clase.
                       // Así aparece en cualquier programa
                       // que quiera utilizar
                       // las propiedades y métodos de un objeto
                       // de tipo Noticia
```

Al ejecutarse esta instrucción se procesa el método constructor definido en la clase Noticia. La función puede tener parámetros que utilizaremos del modo que más nos convenga, por ejemplo, para dar valores iniciales a algunas propiedades. Tal como recordaremos, los parámetros pueden ser opcionales con valores predeterminados:

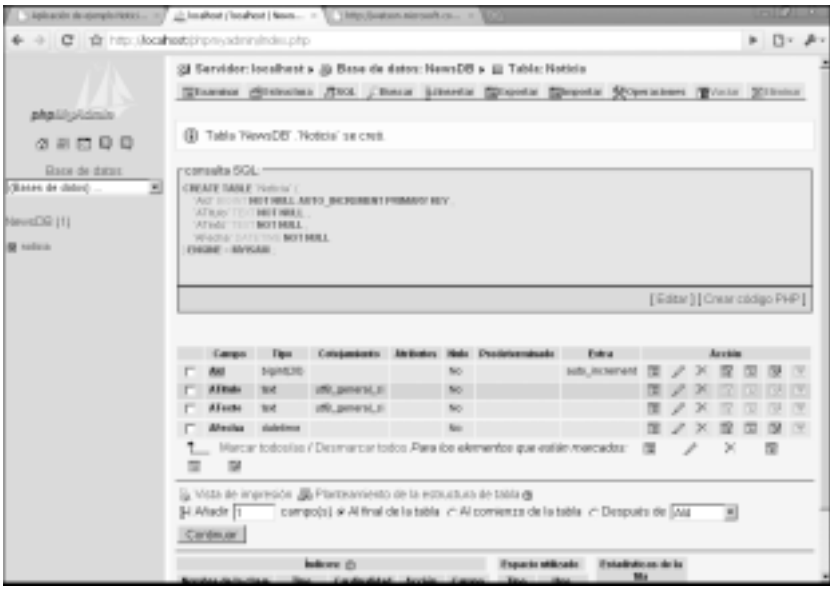
```
function __construct($p1=0,  $p2="", $p3="", $p4= 0) {
    $this->AId = $p1;
    $this->ATitulo  = $p2;
    $this->ATexto   = $p3;
    $this->AFecha   = $p4;
}
```

Ahora podemos crear la tabla siguiendo esta información. En la pantalla de creación de la tabla Noticia se nos pide que indiquemos la cantidad total de campos (columnas) que tendrá la tabla (en este caso, 4):



Al hacer clic en Continuar podremos definir los campos según el criterio establecido y después hacemos clic en el botón Grabar.

En la siguiente pantalla aparece la sentencia SQL que utilizó el asistente para crear la tabla (Create Table) y también se muestra una rejilla que nos permite realizar distintas acciones con los campos (Cambiar, Eliminar, Definir como clave primaria, Crear un índice, Indicar que no se admiten duplicados, etc.).



Esta pantalla también nos permite comenzar a insertar registros.

Entre los archivos de la aplicación se incluye el archivo creartabla.sql que posee las sentencias SQL para crear las tablas sin necesidad de utilizar la pantalla de definición de los campos de una tabla.

Métodos de la clase Noticia

La clase Noticia nos permitirá crear objetos Noticia y, dada la definición del proyecto Noticias, será necesario que esta clase tenga métodos que nos facilite la lectura, inserción, modificación, eliminación y visualización de noticias. Por lo que desarrollaremos los siguientes métodos:

- Leer
- Insertar
- Modificar
- Eliminar
- Visualizar

Siempre que necesitemos leer una noticia podremos hacerlo mediante su método leer() y pasando como parámetro el número de la noticia, por ejemplo:

```
// ejemplo de código que se utilizaría fuera de la clase
// para obtener el título de la noticia número 2
$noticia  = New Noticia();
// lectura de la noticia número 2
$noticia->leer(2);
print $noticia->ATitulo;
```

Por lo tanto, le método leer() debería tener una codificación similar a esta:

```
function leer($numNot) {
    $this->AId = $numNot;

    // si el identificador no es mayor a cero no se accede a la
    // base de datos
    if ($this->AId > 0) {
         // Crear un objeto AuxDB
         $db = new AuxDB();
         $db->conectar();

         // sentencia SELECT para leer una noticia determinada
         $sql = "Select ATitulo, ATexto, AFecha From ";
         $sql.= "Noticia Where AId ='".$this->AId . "'" ;
```

```
        $rst = $db->ejecutarSQL($sql);

        // desconexión con el servidor de base de datos
        $db->desconectar();

        // obtener la fila de datos
        $fila = $db->siguienteFila($rst);
        $this->ATitulo = $fila['ATitulo'];
        $this->ATexto = $fila['ATexto'];
        $this->AFecha = $fila['AFecha'];
    }
}
```

El método insertar() nos permite añadir nuevas noticias a la tabla Noticia. Este método presupone que las propiedades del objeto ya recibieron las asignaciones y sólo le queda insertar el registro en la tabla Noticia. Obsérvese el uso de la función mysql_escape_string que incluye los escapes necesarios en la cadena para que ésta sea apta para ser tratada por la función que ejecute la consulta, por ejemplo, insertando los caracteres de escape en símbolos problemáticos como el apóstrofo. También se emplea la función Now() para actualizar el campo de fecha y hora.

```
function insertar() {
    // Crear un objeto AuxDB
    $db = new AuxDB();
    $db->conectar();

    // sentencia INSERT para insertar una noticia
    $sql = "Insert Into Noticia (ATitulo, ATexto,
         AFecha) Values ('";
    $sql.= mysql_escape_string($this->ATitulo) . "', '";
    $sql.= mysql_escape_string($this->ATexto) . "', ";
    $sql.=  "NOW() ) ";

    // ejecución de la sentencia SQL
    $db->ejecutarSQL($sql);

    // desconexión con el servidor de base de datos
    $db->desconectar();
}
```

La actualización de los campos de una noticia determinada se efectúa mediante el método modificar() y su funcionamiento presupone que la noticia ya existe en la tabla.

Los campos que se actualizan son el título de la noticia, el propio texto o la imagen, ya que el identificador es inmodificable y la fecha se asigna en la inserción y queda fija.

```
 function modificar() {
    // Crear un objeto AuxDB
    $db = new AuxDB();
    $db->conectar();

    // sentencia UPDATE para modificar una noticia
    $sql = "Update Noticia Set ATitulo='" .
         mysql_escape_string($this->ATitulo) . "', ";
    $sql.= "ATexto ='". mysql_escape_string($this->ATexto) ."' ";
    $sql.= "Where AId=' " . $this->AId . "'";

    // ejecución de la sentencia SQL
    $db->ejecutarSQL($sql);

    // desconexión con el servidor de base de datos
    $db->desconectar();
}
```

El editor puede eliminar noticias utilizando el método eliminar(). Al hacerlo también nos ocuparemos de eliminar todos los comentarios relacionados. Algunos gestores de base de datos mantienen la integridad referencial y realizan la eliminación en cascada de modo automático. Al hacerlo manualmente nos aseguramos que este método funciona correctamente en todos los sistemas de gestión de bases de datos.

```
function eliminar($p1) {
    // Crear un objeto AuxDB
    $db = new AuxDB();
    $db->conectar();

         // sentencia DELETE para eliminar todos los comentarios
    $sql = "Delete From Comentario Where CAId=" . $p1 . " ";

    // ejecución de la sentencia SQL
    $db->ejecutarSQL($sql);

    // sentencia DELETE para eliminar una noticia
    $sql = "Delete From Noticia Where AId=" . $p1 . " ";

    // ejecución de la sentencia SQL
    $db->ejecutarSQL($sql);

    // desconexión con el servidor de base de datos
    $db->desconectar();
}
```

La visualización de una noticia es un método que emplearemos cuando necesitemos mostrar en una página todos los datos de una noticia. Tal como se puede observar, se utiliza la función nl2br para convertir nuevas líneas a saltos de línea

de HTML. Recordemos que los campos de título y de texto de la noticia son campos de texto libre que pueden incluir saltos, apóstrofos, comillas y otros caracteres especiales. Los apóstrofos y las comillas pueden crear problemas en las sentencias SQL, tal como hemos visto al desarrollar los métodos insertar y modificar; por su parte, es necesario que el texto se adapte para su presentación en HTML y allí es donde entra en juego la función nl2br (que significa, nueva línea a salto, new line to break).

```
function visualizar() {
    // Fija la información de región
    setlocale(LC_TIME, "sp_SP", "SP");

    // Se genera código HTML con la información de la noticia
    $ret = "<B>" . nl2br($this->ATitulo) . "</B><BR><BR>";
    $ret.=  nl2br($this->ATexto) . "<BR><BR>";
    $ret.=  $this->AFecha . " :: ";

    // Se incluye un enlace para producir comentarios de la noticia
    $colCom = new colComentario($this->AId);

    $ret .= "<a href=\"comentario.php?id=" .$this->AId .
     "\">comentarios (" .$colCom->cantComentarios() . ")</a>";

    // retorno de la cadena que se visualiza
    return $ret;

}
}
```

Clase Comentario

La clase Comentario permitirá crear objetos Comentario que hacen referencia a una noticia determinada. Un comentario no puede existir si no está relacionado con una noticia, pero una noticia podría existir sin comentarios.

Veamos cómo se puede definir en PHP la clase Comentario. Seguiremos la misma metodología que utilizamos con la clase Noticia.

En este caso, crearemos un archivo denominado comentario.php y lo colocaremos en la carpeta Noticias\Objetos.

```
<?php
// Comentario.php (definición de la clase)

class Comentario
{
```

A continuación se definen las variables miembro o propiedades de la clase.

Propiedades de la clase Comentario

Las propiedades de la clase Comentario se corresponden con los campos de la tabla Comentario:

- **CId**: Identificador autoincremento (clave primaria) que nos permite definir de modo único a cada comentario de la tabla. Elegiremos un campo BIGINT, no nulo y de autoincremento.
- **CAid**: Es el identificador de la noticia al que se hace referencia en el comentario. Es el campo AId de la tabla Noticia. Al ser una clave externa: debe existir previamente y no se puede eliminar sin haber eliminado antes todos los comentarios que hagan referencia a esa noticia. Nos preocuparemos por mantener la integridad referencial independientemente del sistema de gestión de base de datos con el que trabajemos, ya que hay algunos que lo hacen automáticamente mediante eliminaciones en cascada y otros no.
- **CUsuario**: Es el nombre del lector que ha realizado el comentario. Lo definiremos como Text y no permitiremos valores nulos.
- **CTexto**: Es el comentario, por lo que normalmente tendrá un texto bastante extenso. Lo definiremos como Text, y no permitiremos que el campo quede con valor nulo (vacío).
- **CFecha**: Es la fecha y hora en la que se realizó el comentario. Se informará siempre por lo que nunca admitirá valor nulo. En realidad es un campo que se informará automáticamente en la sentencia Insert, sin intervención del usuario, de esa manera nos garantizamos su exactitud.

Siguiendo con la definición de la clase en nuestro código php:

```
private $CID;
private $CAId;
Private $CUsuario
private $CTexto;
private $CFecha;

/// Las funciones de acceso nos sirven para recuperar los
// valores de las propiedades
function getCId()        {return $this->CId;}
function getCAId()  {return $this->CAId;}
function getCUsuario()  {return $this->CUsuario;}
function getCTexto()     {return $this->CTexto;}
function getCFecha()     {return $this->CFecha;}
```

Al crearse un ejemplar de la clase Comentario se procesa la función constructora definida en la clase Comentario. La función puede tener parámetros que utilizaremos del modo que más nos convenga, por ejemplo, para dar valores iniciales a algunas propiedades. Tal como recordaremos, los parámetros pueden ser opcionales con valores predeterminados:

```
// método constructor
function __construct($p1=0,  $p2="", $p3="", $p4= 0)
{
    $this->CId = 0;
    $this->CAId = $p1;
    $this->CUsuario = $p2;
    $this->CTexto = $p3;
    $this->CFecha = $p4;
}
```

Ahora podemos crear la tabla siguiendo esta información. En la pantalla de creación de la tabla se nos pide que indiquemos la cantidad total de campos (columnas) que tendrá la tabla (en este caso, 5). Al pulsar el botón Continuar para definir los campos según el criterio establecido y pulsamos el botón Grabar.

En la siguiente pantalla aparece la sentencia SQL que utilizó el asistente para crear la tabla (Create Table) y también se muestra una rejilla que nos permite realizar distintas acciones con los campos (Cambiar, Eliminar, Definir como clave primaria, Crear un índice, Indicar que no se admiten duplicados, etc.). Esta página también nos permite comenzar a insertar registros.

Métodos de la clase Comentario

La clase Comentario nos permitirá crear objetos Comentario y, dada la definición del proyecto Noticias, será necesario que la clase Comentario tenga métodos que nos facilite la lectura, inserción y visualización de comentarios.

No incluimos métodos para modificar y eliminar comentarios porque una vez insertados ya serán inmodificables. El editor o redactor de las noticias no realiza una supervisión previa del contenido de los comentarios antes de hacerlos visibles públicamente.

Es habitual que esta clase de sistemas se encuentren controlados o moderados por el administrador para evitar que comentarios fuera de tono o lugar queden insertados en el sitio. Esto se logra implementando una tercera tabla de comentarios recibidos y no hechos públicos que sólo pueda ser visible por el administrador. Cuando el administrador aprueba su contenido lo libera (transfiere) a la tabla Comentario. En caso contrario puede eliminarlo o dejarlo en esa tabla de visualización restringida al administrador.

En este caso, y por razones de simplificación del ejemplo, nos limitaremos a un sitio web no moderado y simplemente desarrollaremos los siguientes métodos:

- Leer
- Insertar
- Visualizar

Siempre que necesitemos leer un comentario podremos hacerlo mediante su método leer que podría tener una codificación similar a esta:

```
function leer($numCom) {
    $this->CId = $numCom;
     // si el identificador no es mayor a cero no se accede a la
    // base de datos

    if ($this->CId > 0) {
          // Crear un objeto AuxDB
          $db = new AuxDB();
          $db->conectar();

          // sentencia SELECT para leer un comentario determinado
          $sql = "Select CAId, CUsuario, CTexto, CFecha From ";
          $sql.= "Comentario Where CId ='".$this->CId . "'" ;
          $rst = $db->ejecutarSQL($sql);

          // desconexión con el servidor de base de datos
          $db->desconectar();

          // obtener la fila de datos
          $fila = $db->siguienteFila($rst);
          $this->CAId = $fila['CAId'];
          $this->CUsuario = $fila['CUsuario'];
          $this->CTexto = $fila['CTexto'];
          $this->CFecha = $fila['CFecha'];
    }
}
```

El método insertar() nos permite añadir nuevos comentarios a la tabla Comentario. Este método presupone que las propiedades del objeto ya recibieron las asignaciones y sólo le queda insertar el registro en la tabla Comentario. Tal como se hizo en la clase Noticia, aquí también se emplea la función mysql_escape_string que incluye los escapes necesarios en la cadena para que ésta sea apta para ser tratada por la función de consulta SQL, por ejemplo, insertando los caracteres de escape en símbolos problemáticos como el apóstrofo.

```
function insertar() {
    // Crear un objeto AuxDB
    $db = new AuxDB();
    $db->conectar();

    // sentencia INSERT para insertar un comentario
    $sql = "Insert Into Comentario (CAId, CUsuario, CTexto, CFecha)
Values (";
    $sql.=  $this->CAId . ",'" . mysql_escape_string($this->CUsuario)
. "', '";
    $sql.=  mysql_escape_string($this->CTexto) . "', ";
    $sql.=  "NOW() ) ";

    // ejecución de la sentencia SQL
```

```
    $db->ejecutarSQL($sql);

    // desconexión con el servidor de base de datos
    $db->desconectar();
}
```

La visualización de un comentario se obtiene con el método visualizar.

```
function visualizar() {
    // Fija la información de región
    setlocale(LC_TIME, "sp_SP");

    // Se genera código HTML con la información del artículo
    $ret =  "<BR>" . nl2br($this->CTexto) . "<BR><BR>";
    $ret.= "Comentado por:  <B>" . nl2br($this->CUsuario) .
           "</B>  (";
    $ret.= $this->CFecha . ")<BR><BR>";

         return $ret;
}
}
```

Gestión de una colección de noticias: ColNoticias

En muchos momentos de nuestra aplicación nos interesará tratar con la colección completa de noticias en lugar de hacer con una noticia en particular. La clase ColNoticias no representa un registro de la tabla Noticia sino que representa a todo el conjunto de registros de la tabla Noticia. Cuando necesitemos crear una lista de las noticias disponibles nos resultará más sencillo concentrar el código que resuelve esa situación en una clase que representa a toda la tabla.

Nota: Un objeto creado a partir de la clase ColNoticias representa a toda la tabla Noticia.

¿Qué métodos debe poseer esta clase? Su principal misión es mostrar la lista completa de noticias, por lo que tendrá un método que se denominará obtenerLista. Esta lista nos servirá para seleccionar una noticia determinada a fin de modificarla o eliminarla.

También es necesario visualizar la lista detallada de las noticias con su contenido, a este método lo denominaremos visualizarTodos.

Los métodos citados devolverán una cadena con el código HTML que se añade directamente a la página que se está generando.

Además aprovecharemos al método constructor de la clase para que cargue la matriz de objetos Noticia.

Veamos ahora la codificación de la clase ColNoticias, que guardaremos en un archivo con el nombre de la clase y con la extensión .php, es decir, ColNoticias.php y que almacenaremos en la carpeta Objetos.

```
 <?php
// colNoticias.php

class ColNoticias
{
// en esta variable privada se almacenan todos los objetos
// Noticia
private $News = Array();
```

El primer método que desarrollaremos será el constructor de la clase:

```
function __Construct() {

 // Crear un objeto AuxDB
 $db = new AuxDB();
 $db->conectar();

 // sentencia SELECT para leer todas las noticias
 $sql = "Select AId, ATitulo, ATexto, AFecha From ";
 $sql .= "Noticia Order by AFecha Desc" ;
 $rst = $db->ejecutarSQL($sql);
```

Ya tenemos leído el conjunto de resultados de toda la tabla Noticia, ahora debemos realizar un bucle por el conjunto de resultados e ir cargando las noticias en la matriz. Sacamos provecho del constructor parametrizado que hemos creado al desarrollar la clase Noticia. Los parámetros que se pasan para crear el objeto de la clase Noticia son las columnas correspondiente de la fila de la tabla.

```
 while( $fila = $db->SiguienteFila($rst)) {

 // carga de elementos Noticia
 // uso del constructor parametrizado
  $this->News[] = new Noticia( $fila['AId'],
    $fila['ATitulo'], $fila['ATexto'], $fila['AFecha'] );

 }
 // al finalizar de utilizar el conjunto de resultados lo
 // liberamos
 $db->liberarRecursos( $rst );
}
```

Al finalizar la ejecución del método constructor tendremos cargada la matriz privada $News con todas las noticias de la tabla Noticia.

El segundo método que incluiremos en la clase es obtenerLista(). Este método presupone que la matriz de noticias ya se encuentra cargada y por lo tanto no interactúa con la base de datos sino que lo hace directamente con la variable privada. Su misión es crear código HTML con una lista de selección de noticias.

```
function obtenerLista() {
 // producimos el código HTML con la sentencia Select
 // de encabezamiento
 print("<select name =\"IdNot\">\n");
```

Para cada elemento de la matriz añadimos una línea de opción con el identificador y el título de la noticia. La instrucción más adecuada para recorrer una matriz sin preocuparnos por la cantidad de elementos es la instrucción foreach.

```
 foreach( $this->News as $elem ) {
  print("<option value=\"".$elem->getAId()."\">".$elem->getATitulo().
     "</option>\n");
 }

 // cerrar la instrucción select
 print("</select>\n");
}
```

El método visualizarTodos es el encargado de mostrar las noticias en todos sus detalles. Para obtener los detalles nos apoyamos en el método visualizar de la clase Noticia.

```
function visualizarTodos() {
 $str ="";
 // recorremos toda la matriz con foreach
 foreach( $this->News as $elem ) {
  $str .=  $elem->visualizar() . "<br><br>";
 }

 // devuelve toda la cadena
 return $str;
}
```

Finalmente, el método visualizarLista() nos deveuleve una lista de enlaces a cada una de las noticias.

```
function visualizarLista() {
 // recorremos toda la matriz con foreach
 foreach( $this->News as $elem ) {

  $str = substr($elem->GetATitulo(), 0,20);
  print("<a href=\"index.php?AID=".$elem->GetAId()."\">(".
        $elem->GetAfecha() . ") " . $str . "</a><br/>");
 }

}
```

Con esto queda terminada la clase colNoticias.

Gestión de una colección de comentarios: ColComentario

La justificación de la existencia de esta clase es la misma que la que hemos expuesto en la sección anterior. Los métodos de la clase serán los siguientes:

- obtenerLista
- visualizarDetalles
- cargarComentarios

Veamos ahora la codificación de la clase ColComentario, que guardaremos en un archivo con el nombre de la clase y con la extensión .php, es decir, ColComentario.php y que también almacenaremos en la carpeta Objetos, con el resto de las clases.

```
<?php
// colComentario.php

class ColComentario
{
// en esta variable privada se almacenan todos los objetos
// comentario
private $Coms = Array();
```

El primer método que desarrollaremos será el constructor de la clase.

```
function __construct( $inot ) {
    // Crear un objeto AuxDB
    $db = new AuxDB();
    $db->conectar();

    // sentencia SELECT para leer todos los comentarios
    $sql = "Select * From ";
    $sql.= "Comentario Where CAId ='$inot' Order by CFecha Desc" ;
```

```
    $rst = $db->ejecutarSQL($sql);
```

Ya tenemos leído el conjunto de resultados de toda la tabla Comentario de una noticia determinada, ahora debemos realizar un bucle por el conjunto de resultados e ir cargando los comentarios en la matriz. Sacamos provecho del constructor parametrizado que hemos creado al desarrollar la clase Comentario. Los parámetros que se pasan para crear el objeto de la clase Comentario son las columnas correspondientes de la fila de la tabla.

```
    while( $fila = $db->SiguienteFila($rst)) {
          // carga de elementos Comentario
          // uso del constructor parametrizado
          $this->Coms[] = new Comentario( $fila['CAId'],
                $fila['CUsuario'],
                $fila['CTexto'],
                $fila['CFecha']);
    }
    // al finalizar de utilizar el conjunto de resultados lo
    // liberamos
    $db->liberarRecursos( $rst );

}
```

Al finalizar la ejecución del método constructor tendremos cargada la matriz privada $Coms con todos los comentarios de una noticia determinada.

El segundo método que incluiremos en la clase es obtenerLista(). Este método presupone que la matriz de comentarios ya se encuentra cargada y por lo tanto no interactúa con la base de datos sino que lo hace directamente con la variable privada, matriz $Coms. Su misión es crear código HTML con una lista de selección de comentarios.

```
function obtenerLista() {
    // producimos el código HTML con la sentencia Select
    // de encabezamiento
    print("<select name =\"Coms\">\n");
```

Para cada elemento de la matriz añadimos una línea de opción con el identificador y el texto del comentario.

```
foreach( $this->Coms as $elem ) {
          print("<option value=\"".$elem->getCId()."\">".
                $elem->getCTexto()."</option>\n");
    }

    // cerrar la instrucción select
```

```
    print("</select>\n");
}
```

Un método auxiliar nos permite conocer la cantidad de comentarios de una noticia determinada:

```
function cantComentarios() {
    // devuelve la cantidad de comentarios de una noticia
    return count($this->Coms);
}
```

Nos queda un último método, visualizarTodos, encargado de mostrar los comentarios. Para obtener los detalles nos apoyamos en el método visualizar de la clase Comentario.

```
function visualizarTodos() {
    $str ="";
    // recorremos toda la matriz con foreach
    foreach( $this->Coms as $elem ) {
         $str .= $elem->visualizar();
    }

    // devuelve toda la cadena
    return $str;
}

}
```

Con esto queda terminada la clase colComentario y también todas las clases del proyecto. Aunque hemos codificado poco, lo que hemos hecho es un gran avance del proyecto; los proyectos basados en el uso de clases tienen la característica de una codificación más compacta y segura.

Estructura de las páginas PHP

Ya hemos visto en la organización física del proyecto que tendremos varias carpetas en donde distribuiremos los archivos de la aplicación.

La carpeta Objetos nos permite agrupar todas las clases del proyecto. Ahora veremos qué es lo que colocaremos en la raíz principal de la aplicación Noticias y en la carpeta Noticias/Admin.

Nota: Utilizaremos archivos de inclusión para evitar la repetición de código.

En la carpeta raíz del proyecto (Noticias) crearemos estos archivos:

- header.php: Archivo de inclusión que contendrá la definición de las funciones comunes, el encabezamiento HTML de la página y la barra de navegación para las páginas utilizadas por los usuarios normales (no administradores).
- estilos.php: Archivo de inclusión que define los estilos utilizados en las páginas.
- footer.php: Archivo de inclusión que contendrá el cierre de las páginas HTML.
- index.php: Contendrá la página de inicio del sitio para los visitantes (el redactor o administrador utilizará otra página de igual nombre pero en otra carpeta).
- comentario.php: Es la página que permite consultar noticias y añadir comentarios a una noticia determinada.

En la carpeta Admin del proyecto (Noticias/Admin) crearemos estos archivos:

- header.php: Archivo de inclusión que contendrá la definición de las funciones comunes, el encabezamiento HTML de la página y la barra de navegación para las páginas utilizadas por el administrador.
- footer.php: Archivo de inclusión que contendrá el cierre de las páginas HTML.
- index.php: Contendrá la página de inicio del sitio para el administrador.
- agregar.php: Es la página que permite añadir nuevas noticias en la base de datos NewsDB.
- modificar.php: Es la página que permite modificar el texto de las noticias ya publicadas.
- eliminar.php: Es la página que permite eliminar noticias ya publicadas.

header.php

En este archivo de inclusión en la carpeta raíz del proyecto definiremos la función interceptora __autoload() la que se llama automáticamente cuando se intenta crear un ejemplar de una clase que aún no se declaró. Ya la hemos visto en el capítulo dedicado a clases y objetos y aquí se utilizará cada vez que se haga referencia por primera vez a una de las clases del proyecto.

```
<?php
// header.php
```

```
function __autoload($objeto) {
    include("Objetos/".$objeto.".php");
}
?>
<html>
<head>
<title>Aplicación Noticias comentadas capítulo 20</title>
```

Además también incluiremos un archivo que posee los estilos del proyecto.

```
<?php
include("estilos.php")
?>
</head>
<body>
<div align="center">
<table cellpadding="0" cellspacing="0" width="90%">
<tr><td colspan="3" class="título"><B><U>NOTICIAS COMENTADAS</B></
U></td></tr>
<tr><td colspan="3"><br/></td></tr>
<tr>
  <td class="marco" width="40%">
    <B><U>¡ÚLTIMO MOMENTO!</U></B><br/>
     <br/>
```

La parte interesante del archivo header.php que se utiliza en la zona del visitante viene a continuación: de esta manera se genera una lista que se coloca en la parte izquierda de la pantalla para que el lector pueda seleccionar las noticias disponibles. Las noticias se muestras ordenadamente desde la más reciente a la más antigua.

```
<?php
// crea un objeto colección de noticias
$com = New colNoticias();

// visualiza la lista de noticias
// ordenadas por fecha descendente
print( $com->visualizarLista() );
?>
  </td>
  <td width="2%"> </td>
  <td class="marco" width="58%">
```

estilos.php

En este archivo de inclusión codificaremos los estilos que utilizaremos en todas las páginas. Definiremos tres estilos:

- marco
- título
- datos

```
<style type="text/css">
<!--
.marco { border-style: solid;
            font-family: Arial;
            font-size: 14px;
            border-color: blue;
            border-width: 3px;
            vertical-align: top;
            padding:3px;}
.título {font-family: Arial;
              font-size: 18px;
              text-align:center }
.datos { border : thin solid Black;
              font-family : Times;
              font-size : 18px;
              background-color : transparent;
              color: #000000;
}
-->
</style>
```

footer.php

En este archivo de inclusión simplemente cerraremos todas las etiquetas HTML que se han abierto en los archivos previos.

```
  </td>
</tr>
</table>
</div>
</body>
</html>
```

index.php

Este archivo php representa el inicio de la aplicación. Quizá llame la atención su pequeño tamaño pero no hace falta más para presentar en la parte derecha de la pantalla una noticia, si su identificador llega en la URL.

```
<?php
// index.php
```

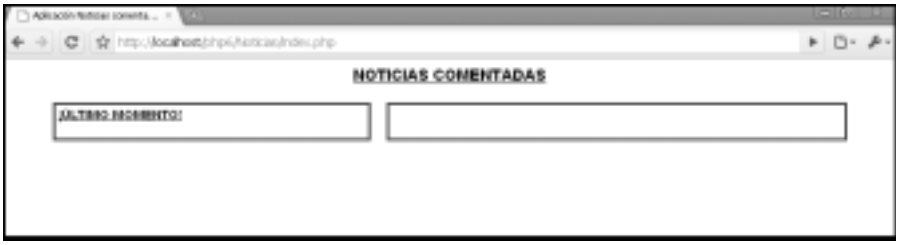
```
include("header.php");

// crea un objeto noticia
$noticia = New Noticia();
$AID = 0;
if( isset( $_GET['AID'] ) ) {
    $AID = $_GET['AID'];
}

if ($AID > 0) {
    $noticia->leer($AID);
    print($noticia->visualizar());
}

include("footer.php");
?>
```

Como ya tenemos codificados todos los archivos y clases que intervienen en esta página podemos comprobar su funcionamiento utilizando nuestro navegador y accediendo a http://localhost/noticias/index.php. Tendríamos que visualizar la siguiente pantalla:



Las bases de datos están vacías y, además, aún no hemos desarrollado la página php para insertar noticias, por lo tanto, por ahora no podemos hacer otra cosa que visualizar está página vacía.

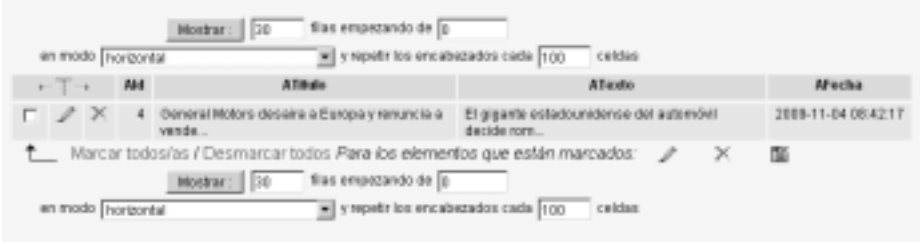
Insertar datos con PhpMyAdmin

Para probar la página con datos podríamos utilizar phpMyAdmin para añadir una fila en la tabla Noticia.

Para ello abrimos la página http://localhost/phpmyadmin/index.php y elegimos nuestra base de datos (NewsDB) en la lista desplegable que aparece a la izquierda de la pantalla. Luego en la línea correspondiente a la tabla Noticia hacemos clic en el icono insertar y en la siguiente pantalla introducimos valores en el campo título, texto, imagen y fecha (recordemos que la clave es de autoincremento). Al finalizar se pulsa en Continuar.

Esto da como resultado la siguiente sentencia SQL:

Ahora podemos pulsar el icono Examinar en la línea correspondiente a la tabla Noticia y veremos que nuestros datos están correctamente introducidos.



Al ya tener una fila en la tabla Noticia podemos probar la pantalla index.php y comprobaremos cómo visualiza las noticias que encuentre en la tabla:

Ahora seguiremos con la pantalla que nos permite consultar una noticia y sus comentarios y también añadir comentarios. A esta página se accede desde el vínculo (comentarios) que se incluye en la página index.php.

Página Comentario.php

Desde la página comentario.php podemos consultar una noticia, visualizar los comentarios que se hayan realizado, ordenados cronológicamente e insertar nuestro propio comentario que se añadirá a la lista. Cualquier lector puede añadir su comentario, la carpeta raíz del proyecto es de libre acceso (ya veremos que eso no ocurre en la zona del administrador).

La página comentario.php utiliza un mecanismo muy habitual de autollamada, es decir, se llama a sí misma.

La primera vez que se ejecuta (cuando se hace clic en el enlace comentarios de la página index.php) el parámetro oper está en blanco. Al estar en blanco, la página simplemente consulta la noticia y la visualiza e inmediatamente después hace lo mismo con los comentarios existentes para dicha noticia y presenta todo en la tabla HTML. Como último paso de esta primera fase genera un formulario que permite añadir los datos de nuestro propio comentario.

Si utilizamos el botón Registrar para grabar el comentario se produce la autollamada antes mencionada. Pero en esta segunda pasada de la página el parámetro oper no está en blanco sino que tiene el valor "comentario", lo que permite ejecutar un código diferente al anterior. Básicamente, en este paso se inserta el comentario en la tabla. Este paso finaliza con otra nueva llamada a la propia página comentario (nuevamente sin parámetro oper) y que se realiza con un temporizador de tres segundos.

Veamos el código detalladamente: en el inicio se comprueba si la matriz global $_GET tiene asignado el elemento oper que, de estar informado, nos llegaría desde el formulario que genera esta misma página.

```
<?php
// comentario.php

include("header.php");

$oper = "";
if( isset( $_GET['oper'] ) ) {
    $oper = $_GET['oper'];
}
```

Esta página tiene dos opciones de proceso:

- **oper=comentario**: Debe insertar el comentario y rellamar a la página.
- **oper=(en blanco)**: Debe visualizar la noticia elegida en la página index.php y todos los comentarios relacionados.

```
switch( $oper ) {
    case "comentario":
```

Antes de insertar un comentario debemos validar que los datos obligatorios se han informado correctamente. En este caso nos conformamos con que los campos texto y título no estén vacíos. Los campos del formulario se encuentran en la matriz $_POST:

```
// campos texto y usuario: deben estar asignados y no vacíos
 if(isset( $_POST['texto']) && !empty( $_POST['texto'] ) &&
   isset( $_POST['usuario']) && !empty( $_POST['usuario'])) {
     // recuperamos el identificador de la noticia
      $idNot = $_POST['id'];

      // se eliminan los escapes
      $_POST['texto'] = stripslashes( $_POST['texto'] );
      $_POST['usuario'] = stripslashes( $_POST['usuario'] );
```

A continuación se crea un objeto Comentario con el identificador del comentario y los dos campos introducidos en el formulario. El identificador del comentario es de autoincremento, por lo cual no se debe informar.

El método constructor del objeto Comentario actualiza las variables miembro del objeto y el método insertar() realiza la inserción física en la tabla Comentario.

```
      $com = new Comentario( $idNot, $_POST['usuario'],
           $_POST['texto'] );
      $com->insertar();
```

Se informa que el comentario quedó registrado en la base de datos y se avisa que en tres segundos se actualiza la página (se vuelve a cargar), lo que hará que también se visualice el comentario que acabamos de añadir.

```
print("Se añadió su comentario.<br/>en 3 segundos se recarga esta
misma página.");
print("<meta HTTP-EQUIV=\"refresh\"
CONTENT=3;URL=\"comentario.php?id=$idNot\">");
}
```

Si la validación de campos resulta negativa se informa el siguiente texto:

```
else
    print("Hay campos faltantes en el formulario del comentario.");
break;
```

En el caso que no se informe el parámetro oper la página debe leer la noticia, generar el código HTML para su visualización (método visualizar). A continuación se crea un objeto colComentario, que tiene todos los comentarios de la noticia tratada (método visualizarTodos).

```
default:
        // si no hay oper = comentario pero hay id.de noticia
         if( isset( $_GET['id'] ) ) {
               $idNot = $_GET['id'];
               // creamos un objeto Noticia
               // y leemos la noticia
               $not = new Noticia();
               $not->leer($idNot);

               // se muestra la noticia como encabezamiento
               print   $not->visualizar() ;

               // y después todos los comentarios registrados
               $com = new colComentario($idNot);

               print "<br><br><b><u>Comentarios:</u></b><br>";
               print( $com->visualizarTodos() );
```

En este punto de la página ya hemos generado el código HTML para visualizar los datos de la noticia y sus comentarios. Ahora queda por crear un formulario de entrada de datos para que el usuario pueda introducir su comentario. Observar el uso del parámetro oper.

```
print("<br>Para añadir su comentario utilice este formulario y pulse
<b>Registrar</b><br/><br/>");
// se crea un formulario con el parámetro oper igual a comentario
```

```
print("<form action=\"comentario.php?oper=comentario\"
method=\"post\">");
print("<table>");
print("<tr><td valign=\"top\">Usuario :</td><td><input type=\"text\"
name=\"usuario\" class=\"datos\"></td>");
print("<td><input type=\"submit\" value=\"Registrar\"></td></tr>");
print("<tr><td valign=\"top\">Comentario dela noticia :</td><td
colspan =\"2\"><textarea name=\"texto\" rows=\"10\" cols=\"65\"
class=\"datos\"></textarea></td></tr>");
print("</table>");
```

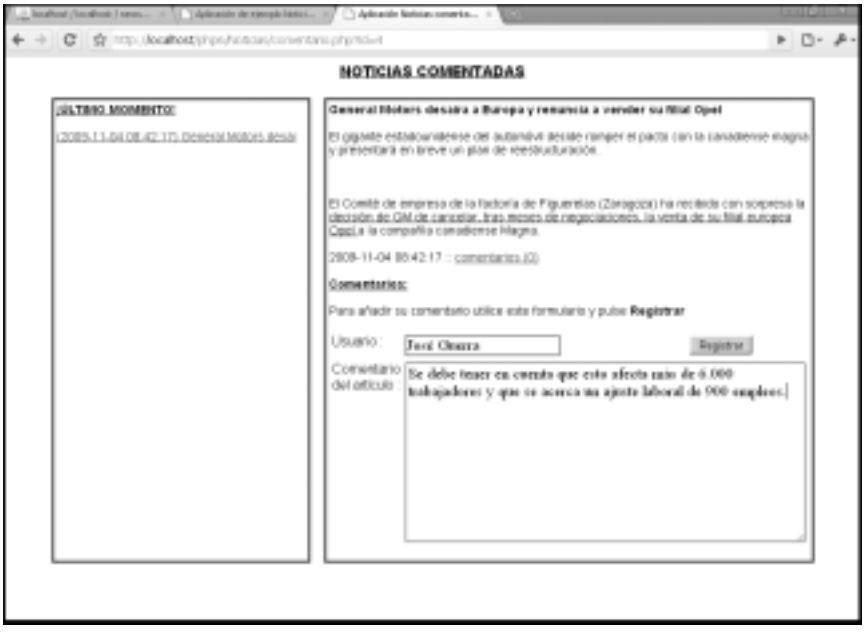
Para no perder el identificador de la noticia se incluye un campo oculto.

```
// utilizamos un campo oculto para pasar el valor del identificador
print("<input type=\"hidden\" name=\"id\" value=\"$idNot\">\n");

print("</form>");
    }
}

include("footer.php");
?>
```

Veamos la página en funcionamento en la inserción de comentarios:



Obviamente, al ser un sitio libre y público puede haber opiniones de todo tipo.

Páginas del administrador y la seguridad

Hasta el momento, en todas las páginas desarrolladas (index.php y comentario.php del directorio raíz de la aplicación Noticias), no nos preocupó limitar el acceso a nadie: cualquier persona puede navegar a nuestro sitio y leer las noticias y sus comentarios, y si lo desea puede hacer su aporte. Pero no sucede lo mismo en las páginas reservadas para el uso del administrador o redactor de las noticias: estas páginas deben tener acceso limitado y no son de uso público: las noticias no pueden ser manipuladas por nadie que no sea el propio redactor, así como tampoco un lector puede modificar los comentarios de otras personas.

Un modo práctico y simple es implementar la seguridad a nivel de directorio. Todos los archivos que se encuentren debajo de ese directorio protegido gozarán automáticamente de la protección de contraseña que Apache implementa con un par de archivos de codificación muy simple: .htaccess y .htpasswd.

Por se motivo al estructurar la organización física del sitio creamos un directorio Admin para las páginas que pueden ser utilizadas sólo por los usuarios autorizados y un directorio Objetos para la definición de las clases. Para este último directorio el acceso debería estar restringido a todos (desde Internet) ya que no posee páginas funcionales, sólo definiciones de clases.

Archivo .htaccess

Este archivo se puede crear con las mismas herramientas de edición que una página php (el Bloc de notas es más que suficiente), ya que se trata de un archivo ASCII. Su nombre lleva a la confusión porque no tiene: en realidad, .htaccess es el nombre de la extensión.

El archivo .htaccess controla al directorio en donde está ubicado y a todos sus subdirectorios. En nuestro caso, Noticias/Admin (que por su parte no tendrá subdirectorios que dependan de él). Se debe tener en cuenta que podemos incluir múltiples archivos .htaccess en los distintos directorios de la aplicación. De ser así, cuando un directorio no tiene su archivo .htaccess, siempre se aplicará la configuración de seguridad establecida en el archivo .htaccess más cercano en el árbol jerárquico.

El archivo .htaccess necesita como complemento un archivo de contraseñas para realizar su tarea. El archivo de contraseñas también carece de nombre y sólo tiene extensión: .htpasswd. También es un archivo ASCII simple (no se debe tratar como binario sino como ASCII) que almacena la lista de nombres de usuario y contraseñas.

```
AuthUserFile "C:\Appserv\www\PHP6\Noticias\ADMIN\.htpasswd"
AuthName ZonaAdmin
AuthType Basic
require valid-user
```

También se podría codificar un nombre de usuario, por ejemplo:

```
require user josé
```

La codificación de este archivo es simple:

- **AuthUserFile**: Ruta de acceso al archivo .htpasswd (podría, y convendría, estar fuera dentro del sitio web)
- **AuthName**: Nombre asignado a la zona protegida. Este nombre saldrá en el cuadro de inicio de sesión.
- **AuthType Basic**: Tipo de autenticación HTTP. La otra alternativa sería `AuthType Digest`.
- **require user**: Usuario que se espera que acceda al sitio. En el caso de que haya más de un usuario se utiliza `require valid-user`.

El archivo `.htpasswd` podría estar en cualquier carpeta del disco del servidor web pero no necesariamente visible desde la web, es decir, una carpeta independiente, por ejemplo: `D:\ carpetasegura\`. Es más, se recomienda colocar el archivo `.htpasswd` fuera de la carpeta del sitio web.

Archivo .htpasswd

Tal como hemos dicho previamente, a diferencia de lo que ocurre con el archivo .htaccess, que está en el propio directorio que controla, o en un nivel superior dentro de la jerarquía, el archivo `.htpasswd` puede estar en el mismo directorio que `.htaccess`, en otro directorio del sitio o incluso en un directorio no visible desde la web (es decir, fuera del conjunto de directorios del sitio web).

La ubicación real del archivo .htpasswd queda establecida en el archivo .htaccess.

La codificación de este archivo es muy simple:

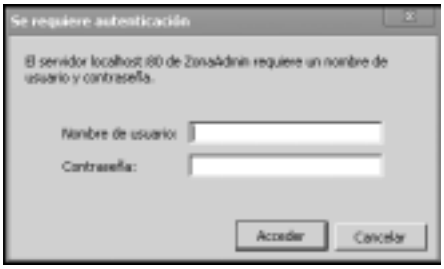
```
nombreUsuario1:contraseña1
nombreUsuario2:contraseña2
...
```

En nuestro caso crearemos el siguiente archivo de usuarios y contraseñas:

```
admin:pswadmin
josé:66R4aSn8tK4Bg
```

Seguridad en acción

Veamos qué sucede cuando dirigimos nuestro navegador a una página que se encuentre dentro de la carpeta Admin:



Tal como hemos codificado nuestro archivo .htpasswd, sólo podrán entrar los usuarios admin y josé, y si aciertan con la contraseña establecida en cada caso.

Seguridad para la carpeta Objetos

La carpeta Objetos posee la definición de las clases del proyecto y no queremos que nadie acceda a esta carpeta desde http. Por lo tanto, en este caso, el archivo .htaccess que colocaremos en este directorio será el siguiente:

```
deny from all
ErrorDocument 403 "<body bgcolor=#ffffff><h1>Acceso prohibido a esta
carpeta</h1>
```

Hemos codificado un mensaje de error personalizado para cuando se presente el error 403 (error de autorización). Y esto es lo que ocurre si queremos acceder a cualquier archivo de esta carpeta utilizando un navegador:

Agregar nuevas noticias

La página agregar.php es la que permite añadir nuevas noticias y es una de las páginas que están almacenadas en la carpeta Admin.

En esta página también utilizamos el mecanismo de autollamada mediante un formulario. La primera vez que se llama a esta página agregar.php, el parámetro oper estará sin asignar y por lo tanto simplemente se presenta un formulario vacío.

Si llenamos el formulario con los datos de la noticia y pulsamos el botón Registrar se producirá la autollamada mencionada, pero en este caso con el parámetro oper con el valor "agregar".

En esta segunda ejecución de la página se realiza la validación de los campos y, si no hay errores, se inserta la noticia en la tabla.

En realidad, la estructura de esta página es muy similar, si no idéntica, a la página comentario.php que hemos visto anteriormente en este proyecto: ambas son funcionalmente páginas de entrada de datos e inserción de registro en una tabla.

La codificación de la página es la siguiente:

```
<?php
// agregar.php

include("header.php");

$oper = "";
if( isset( $_GET['oper'] ) ) {
    $oper = $_GET['oper'];
}

// esta página se llama de dos maneras:
// primero con oper en blanco
// segundo con oper=agregar
switch($oper) {
    case "agregar":
```

En la validación no se permite que los campos tema d la noticia (título) y texto hayan quedado sin informar.

```
    if(isset($_POST['título']) && !empty($_POST['título']) &&
      isset( $_POST['texto'] ) && !empty( $_POST['texto'] ) ) {
              // detecta uso de escape
              $_POST['título'] = stripslashes( $_POST['título'] );
              $_POST['texto'] = stripslashes( $_POST['texto'] );
```

```
            // crea un objeto Noticia
            $not = new Noticia( 0, $_POST['título'],
                  $_POST['texto'] );

            // lo inserta en la tabla
            $not->insertar();

            print("Noticia añadida a la base de datos.");
      }
      else
            print("Faltan datos en el formulario de entrada");

      break;
```

Éste es el código que se ejecutará cuando el parámetro oper no tenga el valor "agregar":

```
   default:
      // la primera vez que se llama a esta página oper estará
      // en blanco
      // se crea un formulario con el parámetro oper = agregar
      // y que llama a esta misma página
      print("Formulario de entrada de noticias : <br/><br/>");
      print("<form action=\"agregar.php?oper=agregar\"
            method=\"post\">");
      print("<table>\n");

      print("<tr><td valign=\"top\">Tema de la noticia:
            </td><td><input type=\"text\" name=\"título\"></td>");
      print("<td><input type=\"submit\" value=\"Registrar\">
            </td></tr>\n");
      print("<tr><td valign=\"top\">Texto :</td>
            <td colspan=\"2\">
            <textarea name=\"texto\" rows=\"20\" cols=\"50\">
            </textarea></td></tr>\n");

      print("</table>\n");
      print("</form>");
}

include("footer.php");
?>
```

Cuando desde la barra de navegación pulsemos la entrada Añadir noticia veremos la siguiente página:

Modificación de noticias

La página de modificación de noticias es la más compleja de las tres páginas de actualización. La implementación de la página la realizaremos en el archivo modificar.php que colocaremos también en la carpeta Admin.

Si bien la estructura es similar a la de la página de inserción (se utiliza un formulario que realiza una autollamada) se añade un paso previo en el que se muestra una lista de selección para elegir la noticia que se quiere modificar. Después de la elección de la noticia se realiza la presentación del contenido en un formulario en el que se permite la modificación del título y del texto. La fecha de alta no se modifica.

Veamos la codificación:

```
<?php
// modificar.php

include("header.php");

// la primera vez que se entra a esta página
// la acción está sin asignar
$oper = "";
```

En la página agregar.php teníamos dos opciones: oper sin asignar y oper igual a "agregar". Aquí hay tres opciones:

- **oper sin asignar**: Presenta la lista de noticias.
- **oper "obtener"**: Presenta el formulario de la noticia para que pueda ser modificada.

- **oper "modificar"**: Se validan los datos introducidos y se realiza la actualización de la tabla.

```
// si ya entró anteriormente la acción puede ser
// obtener
// o
// modificar
if( isset( $_GET['oper'] ) ) {
    $oper = $_GET['oper'];
}

switch($oper) {
    case "obtener":
    // en esta acción se busca la noticia elegida en la primera
    // llamada a la pantalla modificar
         $not = new Noticia();
```

En $_POST['IdNot'] se recibe el identificador de la noticia que se eligió modificar en la lista de selección:

```
         $not->leer($_POST['IdNot']);

         // crea un formulario que llama a esta misma página pero
         // con un parámetro oper igual a modificar
         print("Datos de la noticia: <br/><br/>");
         print("<form action=\"modificar.php?oper=modificar\"
               method=\"post\">");
         print("<table>\n");
         print("<tr><td valign=\"top\">Tema de la noticia :
               </td><td><input type=\"text\" name=\"título\"
               value=\"".$not->getATitulo()."\"></td>");
         print("<td><input type=\"submit\" value=\"Validar\">
               </td></tr>\n");
         print("<tr><td valign=\"top\">Texto :</td>
               <td colspan=\"2\"><textarea name=\"texto\" rows=\"20\"
                cols=\"50\">".$not->getATexto()."</textarea></td>
               </tr>\n");

         print("</table>\n");

         // se crea un campo oculto con el número de noticia
         print("<input type=\"hidden\" name=\"IdNot\"
               value=\"".$_POST['IdNot']."\">\n");
         print("</form>");
         break;
```

En el paso de modificación se realiza la validación de los campos del formulario (no se aceptan campos vacíos) y si se detecta algún error se emite un mensaje que indica que el formulario tiene errores:

```
case "modificar":
    // validación de los campos del formulario
         if(isset($_POST['título']) && !empty($_POST['título']) &&
             isset($_POST['texto']) && !empty($_POST['texto']))     {
              // detecta uso de escape
              $_POST['título'] = stripslashes( $_POST['título'] );
              $_POST['texto'] = stripslashes( $_POST['texto'] );

              // crea un objeto Noticia
              $not = new Noticia( $_POST['IdNot'], $_POST['título'],
               $_POST['texto'] );

              // modifica los datos en la tabla
              $not->modificar();

              print("Noticia modificada.");
         }
         else
              print("Error de validación en datos del formulario");

         break;
```

Pese a que esta es la opción final de la página es la primera que se ejecuta (cuando el parámetro oper está sin asignar). En esta opción se utiliza la clase colNoticias para generar una lista completa de las noticias en la tabla.

```
    default:
    // esta acción se emplea la primera vez que se carga la página
         print("Seleccionar el tema que se quiere modificar :
              <br/><br/>");
         // crea un formulario que llama a la propia página pero
         // con un parámetro oper diferente
         print("<form action=\"modificar.php?oper=obtener\"
              method=\"post\">");

        // crea un objeto colNoticias
         $nots = new colNoticias();

         // genera una lista de selección con los nombres de los
         // temas de las noticias
         $nots->obtenerLista();

         print("<input type=\"submit\" value=\"Modificar\">\n");
         print("</table>\n");
         print("</form>");
}

include("footer.php");
?>
```

Cuando desde la barra de navegación pulsemos la entrada Modificar noticia veremos la siguiente página:

Al elegir una noticia y pulsar el botón Modificar se visualiza el contenido de la noticia (es la misma página modificar.php en su opción "obtener") y podemos cambiar la noticia a voluntad (salvo dejar los campos vacíos). En este caso se modifica la noticia agregada en el paso anterior.

Eliminación de noticias

La página de eliminación de noticias es la más simple de las tres páginas de actualización. La implementación de la página la realizaremos en el archivo eliminar.php que colocaremos también en la carpeta Admin.

El único detalle que merece especial atención es que cuando se elimina una noticia, previamente le eliminamos todos los comentarios para que no queden en la tabla sin ninguna relación. Esto se podría implementar automáticamente con integridad referencial pero algunos sistemas de gestión de base de datos no tienen la funcionalidad de eliminación en cascada y por eso lo implementamos manualmente con una sentencia Delete en la tabla Comentario.

Esta página tiene dos fases:

- **oper sin asignar**: Es primera llamada y presenta la lista de las noticias existentes en la tabla Noticia.

- **oper "eliminar"**: Elimina la noticia elegida. En el método eliminar del objeto Noticia se realizan dos sentencias Delete.

Veamos la codificación:

```
<?php
// eliminar.php

include("header.php");

$oper = "";
if( isset( $_GET['oper'] ) ) {
    $oper = $_GET['oper'];
}
```

Cuando se realiza la autollamada a la página el parámetro oper tiene el valor "eliminar". Simplemente se utiliza el método eliminar de la clase Noticia sin necesidad de crear un objeto y el identificador de la noticia se pasa como parámetro al método.

```
switch($oper) {
    case "eliminar":
         // en la segunda llamada a la página el parámetro oper
         // es eliminar y el valor de $_POST['IdNot'] es el
         // identificador de la noticia que se quiere eliminar
         // crea un objeto Noticia
         $not = new Noticia( $_POST['IdNot']);
         $not->eliminar($_POST['IdNot']);

         print("Se eliminó la noticia");

         break;
```

Cuando se realiza la primera llamada a la página desde el enlace Eliminar noticia el parámetro oper no se informa, por lo que entra en el caso default, que simplemente llena la lista de selección (tal como se realizó en la página modificar.php).

```
default:
         print("Seleccionar un tema que se eliminará de la tabla de
          noticias<br/><br/>");
         // crea un formulario que llama a esta misma página, pero
         // con oper = eliminar
         print("<form action=\"eliminar.php?oper=eliminar\"
              method=\"post\">");

         // crea un objeto colNoticias
         // con la colección de todos las noticias
```

```
        $nots = new colNoticias();

        // crea una lista de selección de temas de las noticias
        $nots->obtenerLista();

        print("<input type=\"submit\" value=\"Eliminar\">\n");
        print("</table>\n");
        print("</form>");
}

include("footer.php");
?>
```

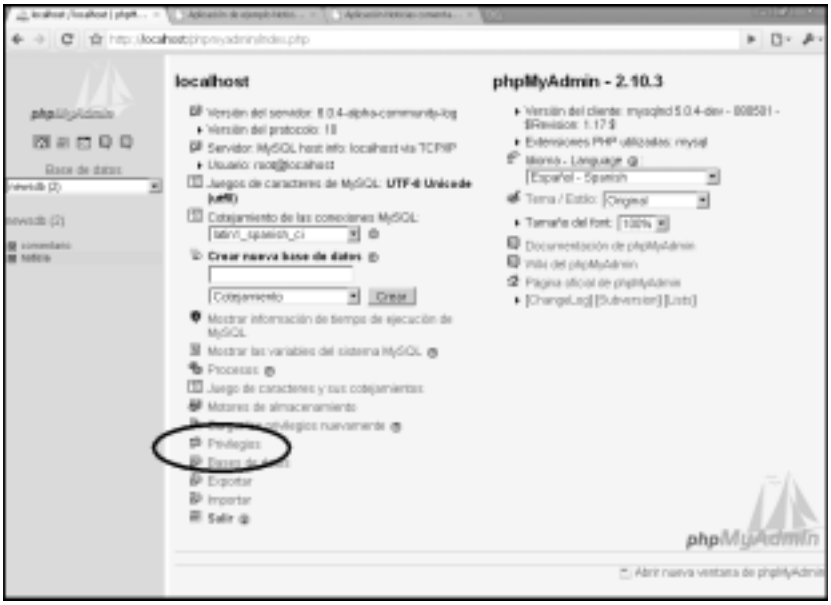
Cuando desde la barra de navegación pulsemos la entrada Eliminar noticia veremos la siguiente página:

Al elegir una noticia y pulsar el botón Eliminar se realiza la autollamada a la página eliminar.php (esta vez con oper igual a "eliminar") y se elimina la noticia y todos los comentarios que pudiera tener.

Gestión de privilegios con PhpMyAdmin

La primera pantalla nos puede mostrar una advertencia que debemos tener en cuenta. En este caso hemos omitido configurar una contraseña para la cuenta privilegiada de MySQL (root) y eso crea una vulnerabilidad muy seria en un entorno de producción. En el entorno de desarrollo podríamos trabajar sin seguir esta recomendación, pero en producción sería un riesgo que no se debe asumir jamás.

Para corregir esta omisión entraremos en la página Privilegios que nos muestra la vista global de los privilegios asignados:



Al pulsar en Privilegios aparece esta página:

En esta página podemos definir las contraseñas para los distintos usuarios y los privilegios globales. Esta página nos permite añadir usuarios y asignarles privilegios. Al hacer clic en el icono Editar los privilegios podemos modificar los privilegios asignados a un usuario

Nota: las contraseñas más difíciles de detectar son las que tienen al menos 8 caracteres y con una combinación de letras mayúsculas, minúsculas, algún carácter especial y algún número. Como no hay nada perfecto, también hay que reconocer que son las más difíciles de recordar.
