Diseño avanzado de bases de datos relacionales

versión: 2023.3

9 de febrero de 2023

Índice

1. Introducción a MySQL-XAMPP

- MySQL `http://www.mysql.com/` es un sistema gestor de bases de datos relacional.
- Es el SGBD de código abierto más conocido a nivel mundial.
- El acceso a las bases de datos de MySQL se realiza a través de una aplicación cliente:
  - Consola de MySQL (instalada con el propio SGBD).
  - MySQL Workbench.
  - phpMyAdmin (XAMPP).
  - Otros.
- Trabajando con la consola de MySQL:
  - Se ejecuta con:

```
mysql -u <nombre_usuario> -p
```

  - Mostrar las bases de datos disponibles:

```
mysql> SHOW DATABASES;
```

  - Cambiar la base de datos actual:

```
mysql> USE <nombre_db>;
```

- Ejecutar una sentencia SQL sobre la base de datos actual:

```
mysql> SELECT DNI,Nombre FROM Estudiantes;
mysql> INSERT INTO Estudiantes VALUES (...);
mysql> CREATE TABLE Asignaturas (
...
);
```

- Ejecutar un script con sentencias SQL:

```
mysql> source nombre_archivo.sql;
```

- La administración de una base de datos mediante sentencias SQL es demasiado tediosa.
- Existen clientes que permiten crear los esquemas relacionales de una base de datos a través de una interfaz gráfica.
- *phpMyAdmin* es una herramienta para administrar bases de datos MySQL a través de un navegador web.
- *phpMyAdmin* necesita Apache + PHP para funcionar.
- XAMPP es una distribución que incluye Apache, MariaDB, PHP y la herramienta *phpMyAdmin*.
- Es gratuito y se puede encontrar en `http://www.apachefriends.org/en/xampp.html`.

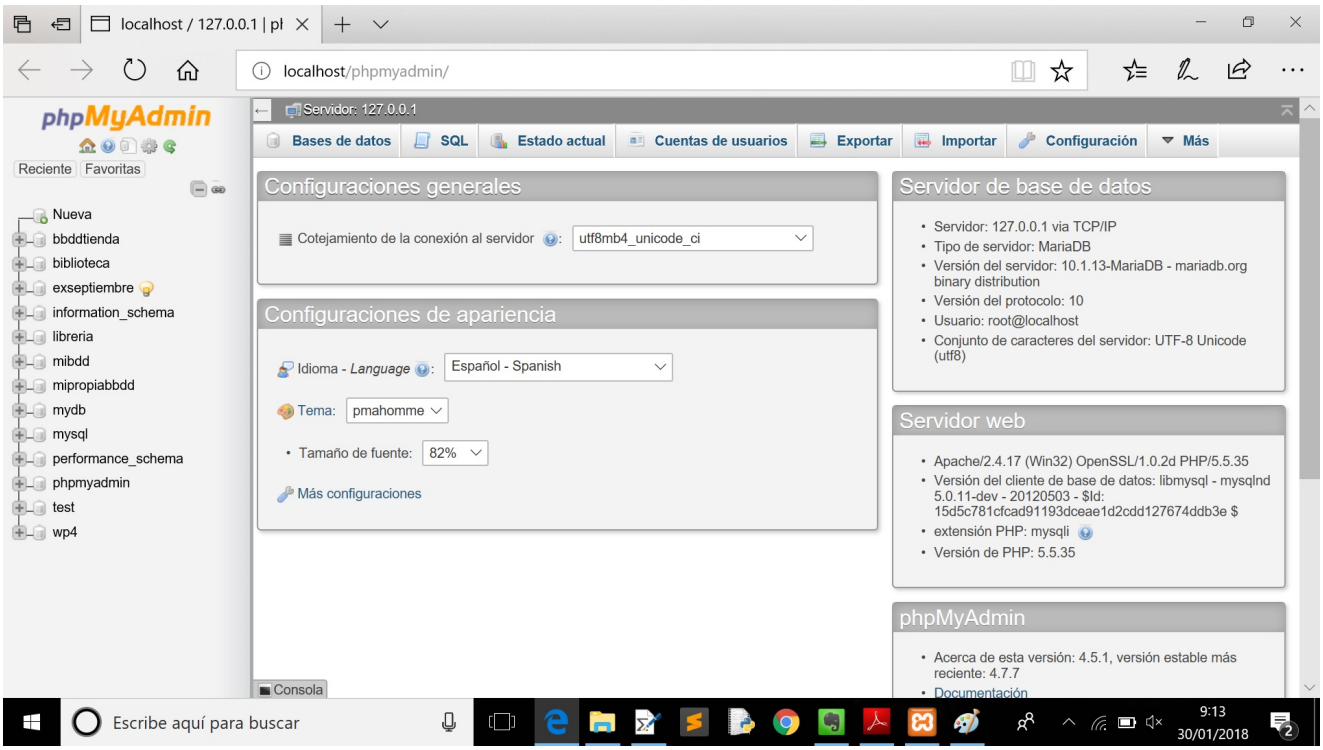

Figura 1: Ventana principal de XAMPP

1.1. XAMPP

- Se especifica el nombre de la base de datos.
- El "cotejamiento" es opcional. Contiene el conjunto de reglas de comparación y ordenación del texto en la base de datos (depende de cada idioma).

2. Control de acceso en SQL

- El lenguaje SQL se compone de:
  - *DDL* (*Data definition language*): `CREATE, DROP, ALTER ...`
  - *DML* (*Data manipulation language*): `SELECT, INSERT, UPDATE, DELETE, CALL...`
  - *DCL* (*Data control language*): `GRANT, REVOKE`
  - *TCL* (*Transaction control language*): `COMMIT, ROLLBACK, SAVEPOINT...`
- Los SGBD permiten definir permisos sobre el tipo de operaciones que pueden realizar los usuarios.
- Si un usuario intenta realizar una operación para la que no tiene permiso, el SGBD la rechazará.
- El estándar de SQL define los siguientes permisos:
  - Leer datos (SELECT)
  - Insertar datos (INSERT)

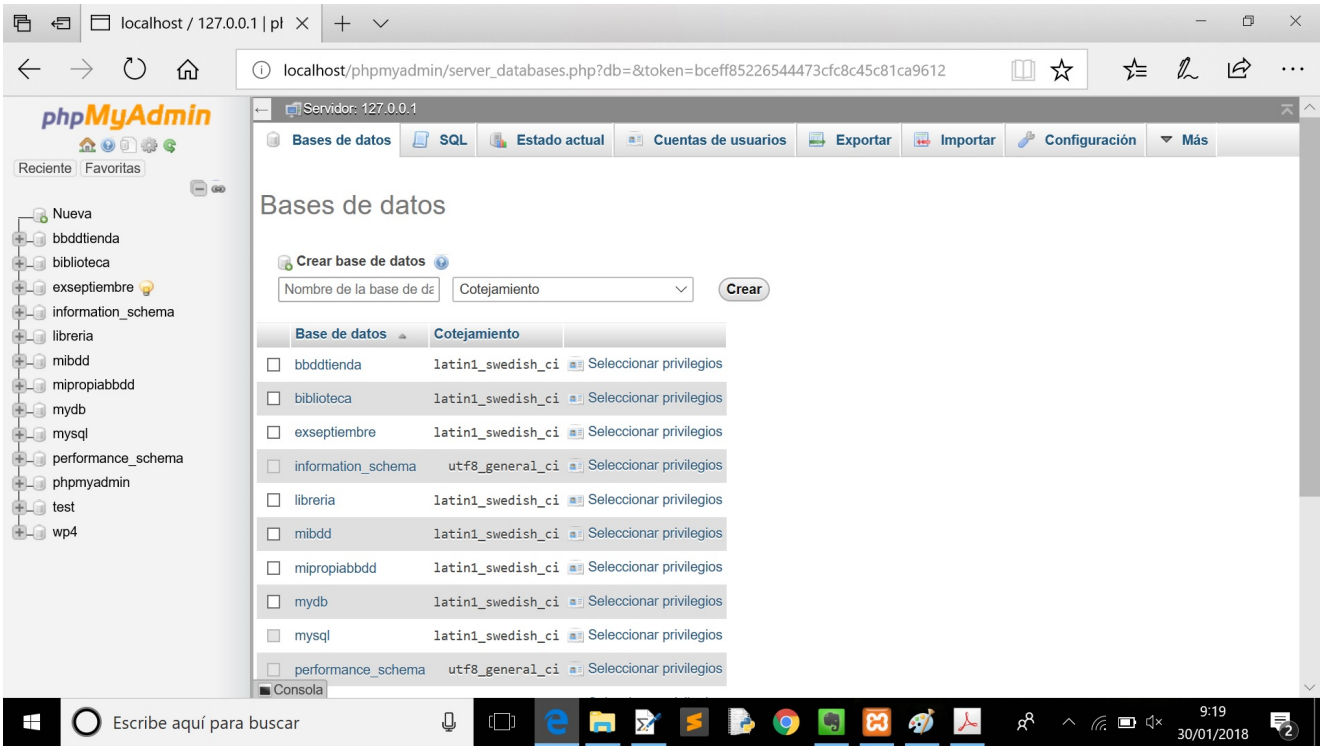

Figura 2: Creando una base de datos con XAMPP

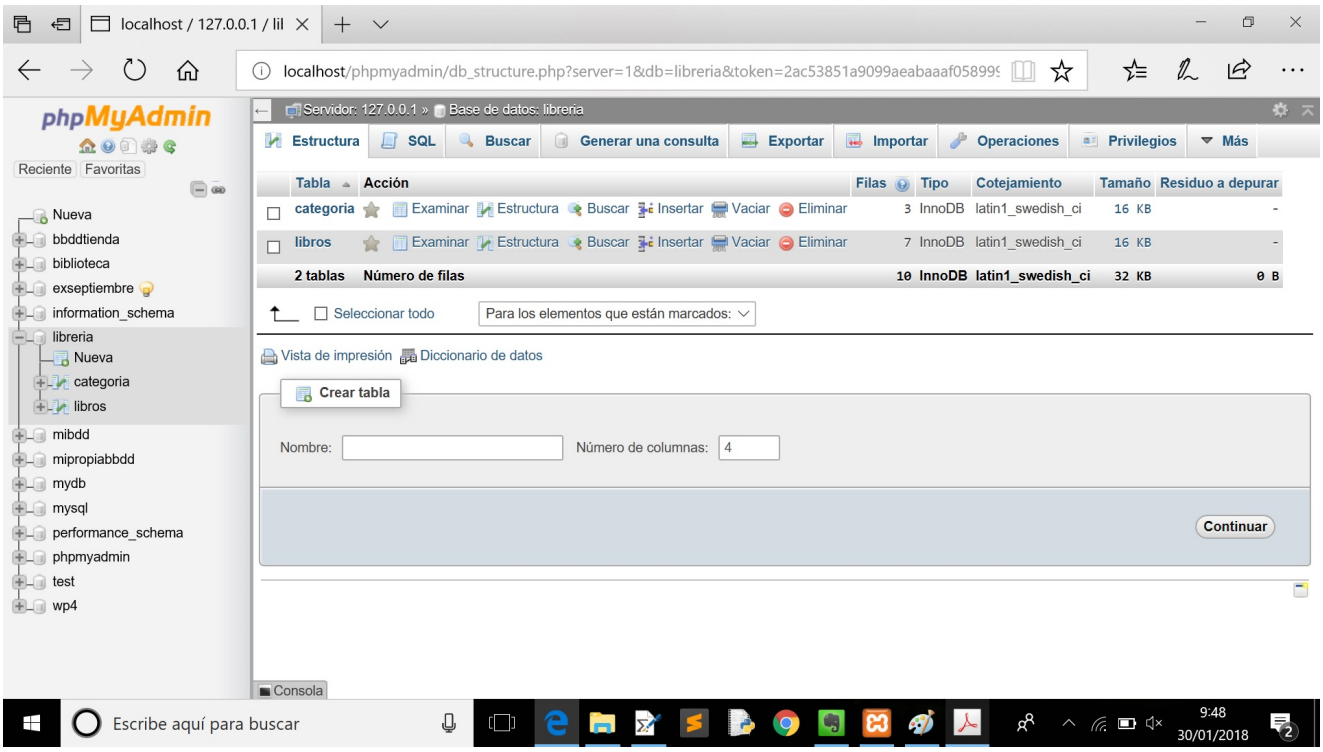

Figura 3: Contenido de la base de datos *libreria*

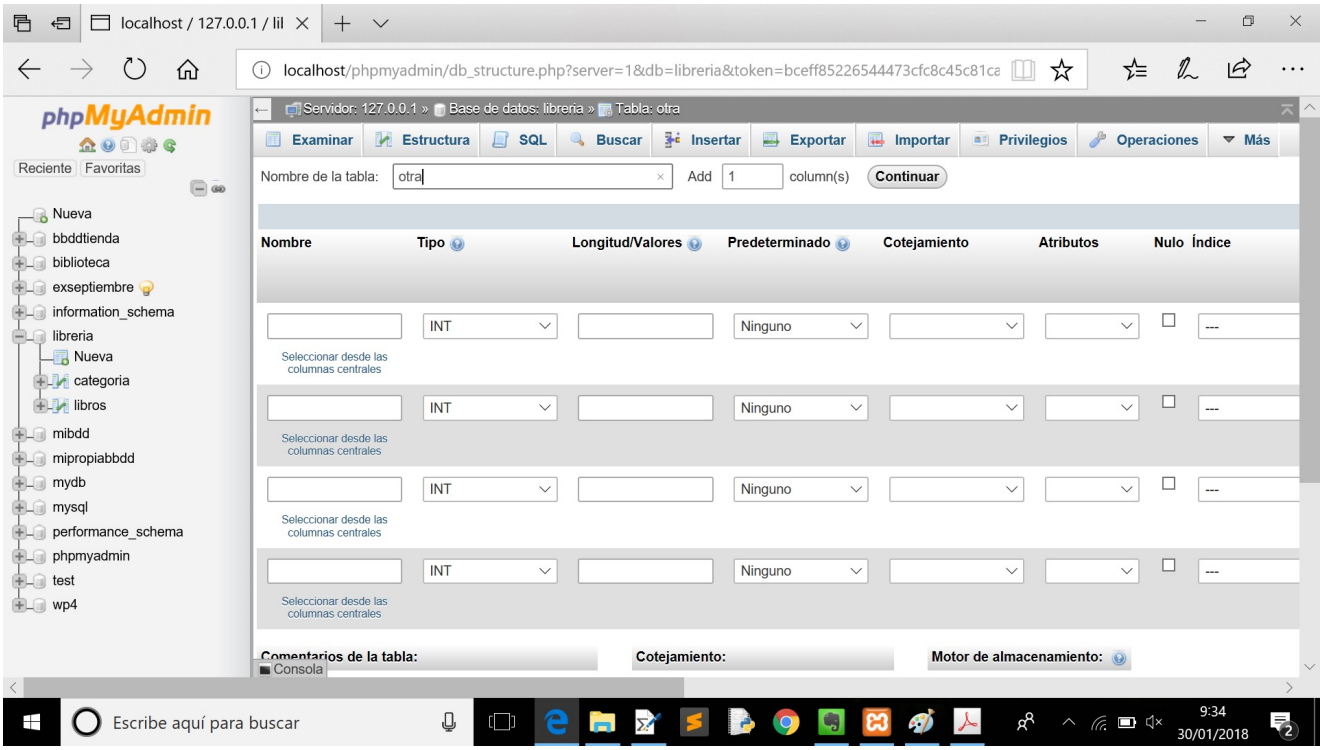

Figura 4: Definiendo la estructura de una tabla *categoria*

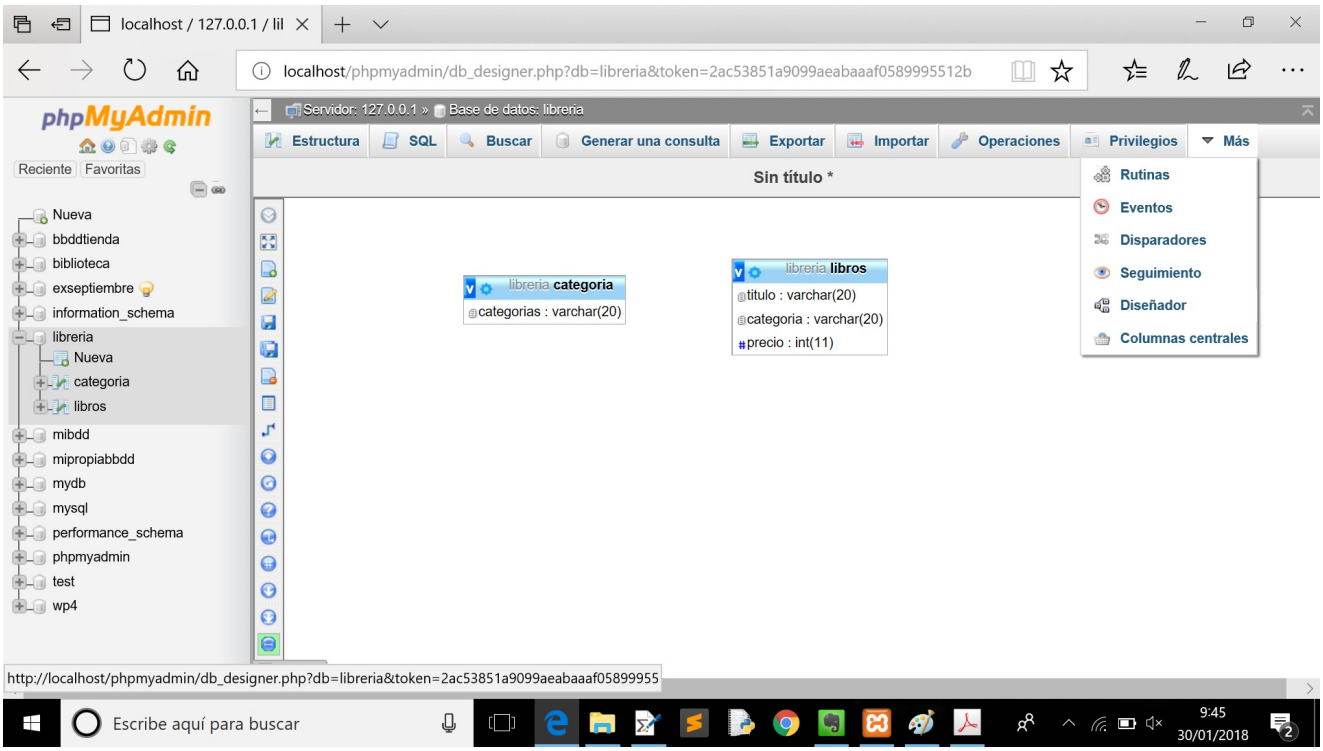

Figura 5: Definiendo relaciones entre tablas *categoria*

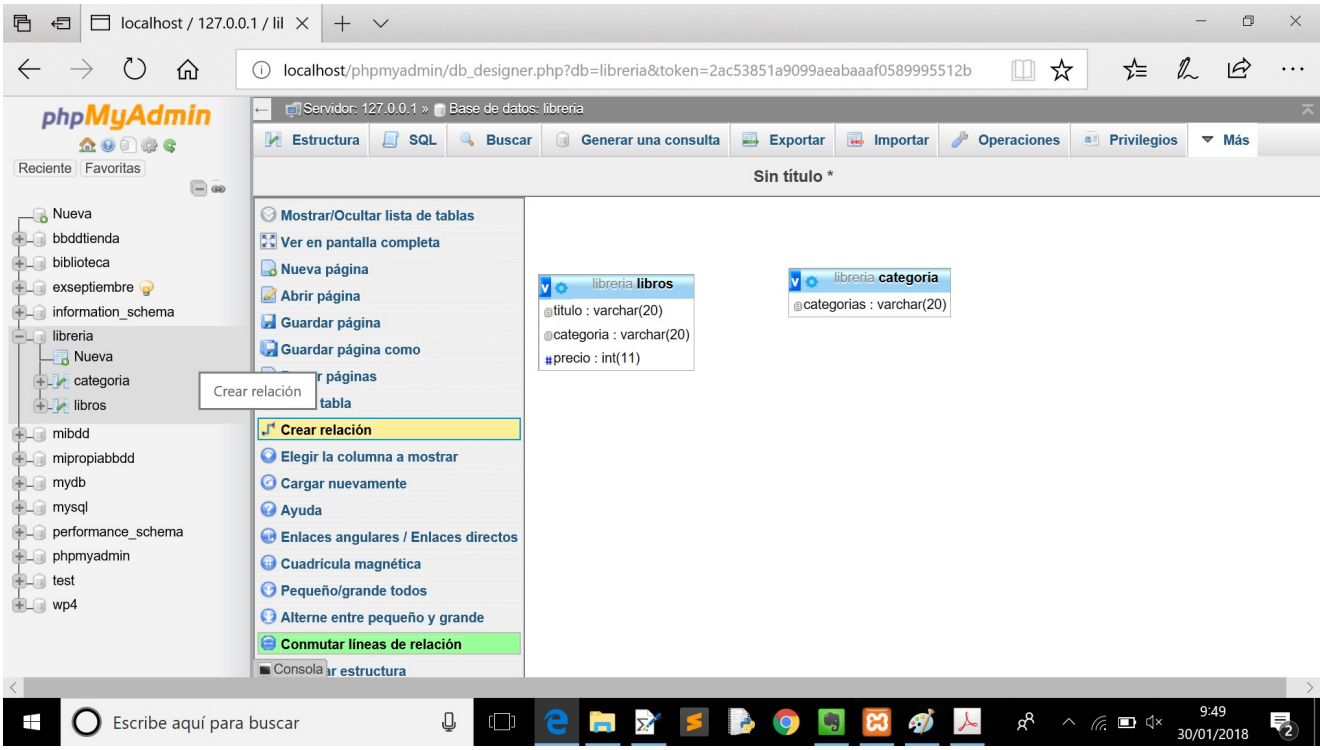

Figura 6: Definiendo relaciones entre tablas *categoria*

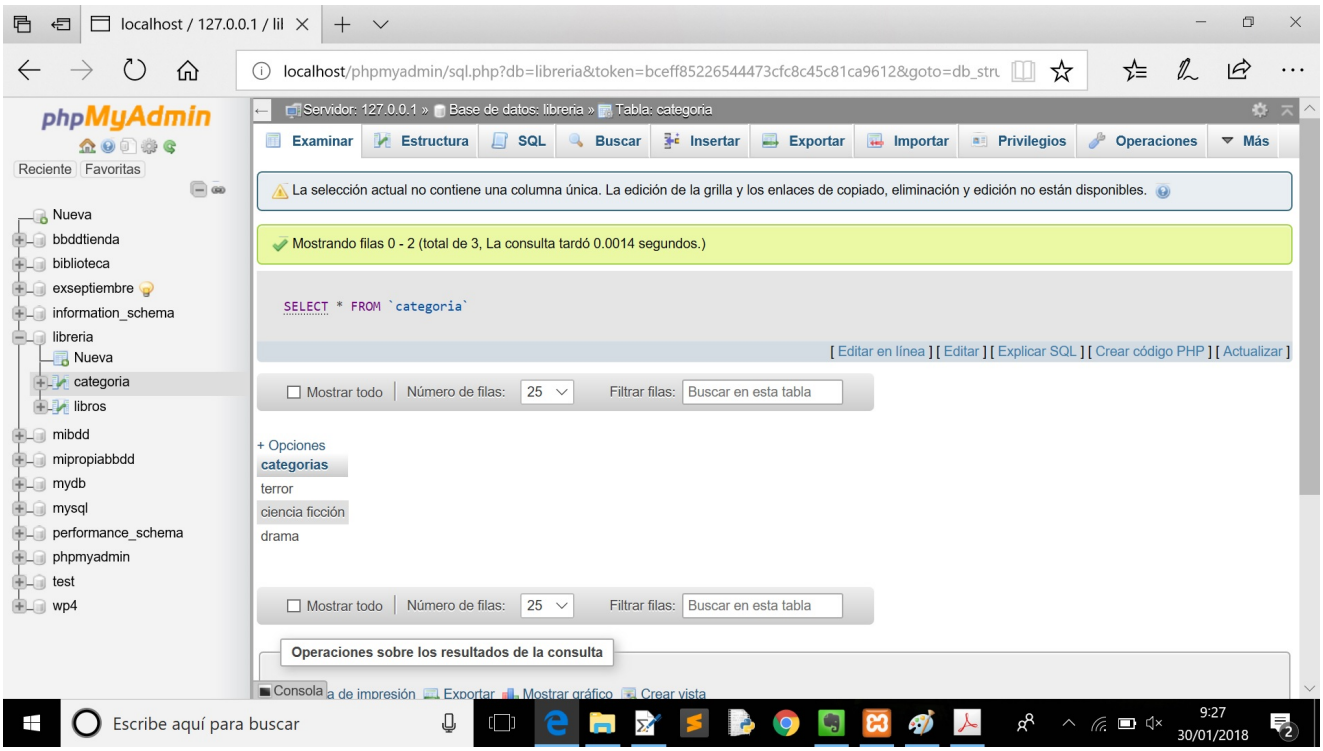

Figura 7: Contenido de la tabla *categoria*

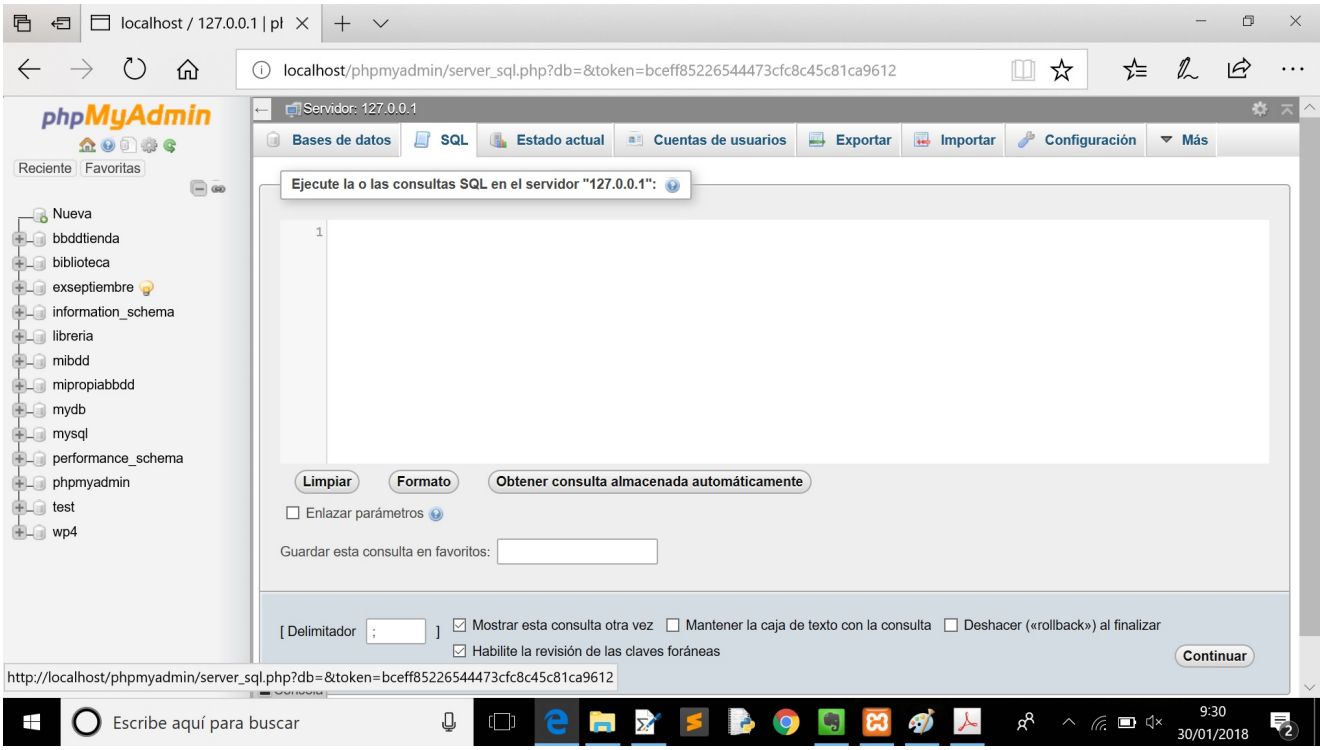

Figura 8: Modo consola de XAMPP *categoria*

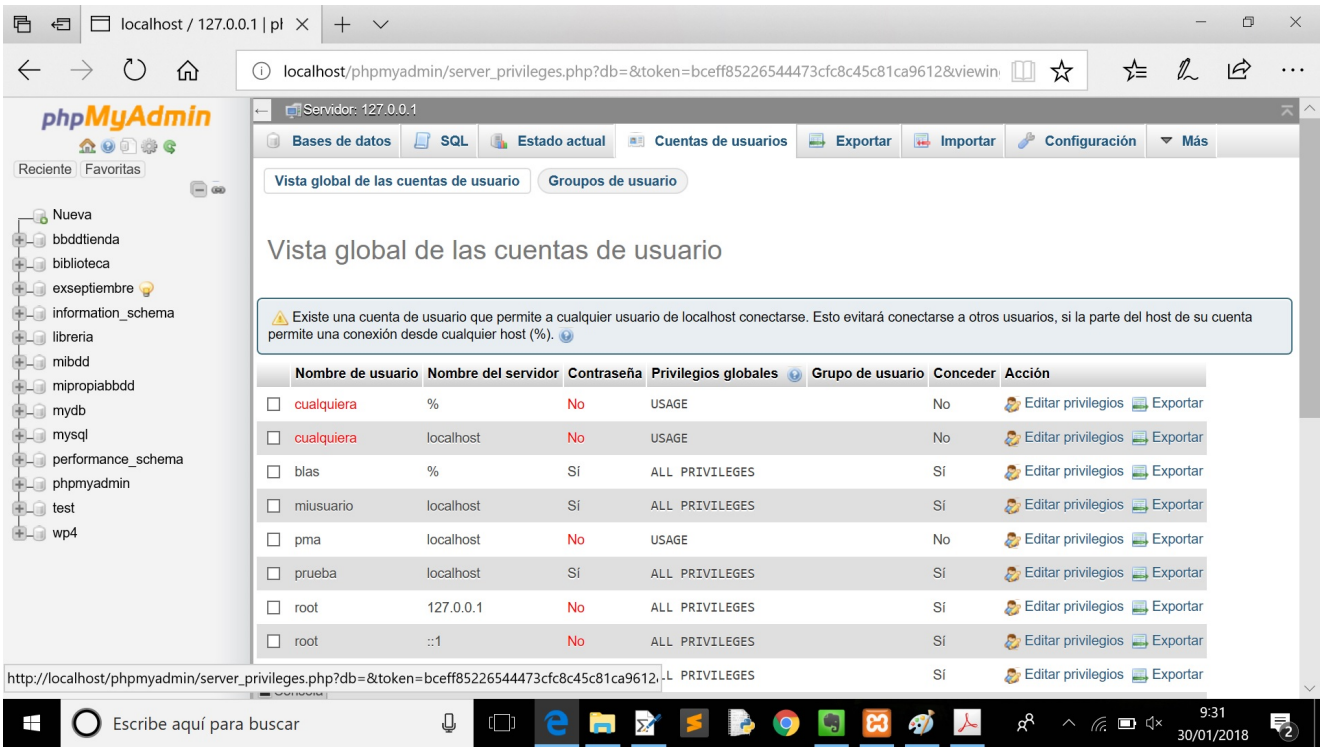

Figura 9: Gestión de usuarios *categoria*

    - Actualizar datos (UPDATE)
    - Borrar datos (DELETE)

- Existe un usuario especial (administrador) que tiene acceso total a las bases de datos almacenadas y puede asignar permisos a los demás usuarios.
- En MySQL/XAMPP, recibe el nombre de `root` y no tiene contraseña asignada por defecto.
- Crear y eliminar usuarios:
    - Según el estándar de SQL, al conceder un permiso a un usuario que no existe éste se crea implícitamente.
    - En MySQL se permite (y se recomienda) crear un usuario explícitamente mediante `CREATE USER`.

```
CREATE USER 'NombreUsuario' IDENTIFIED BY 'Contraseña'
```

    - Para eliminar un usuario se utiliza la cláusula `DROP USER`.

```
DROP USER 'NombreUsuario'
```

- Gestión de permisos:
    - Concesión de permisos. En el estándar SQL se utiliza la cláusula `GRANT`.

```
GRANT Privilegio1, Privilegio2, ...
ON NombreTabla
TO Usuario1, Usuario2, ...
```

      donde cada privilegio puede ser:
        - `SELECT`
        - `INSERT`
        - `UPDATE`
        - `ALL PRIVILEGES`
        - Y muchos más.
    - Conceder permisos:
        - Ejemplo:

```
CREATE USER 'Manuel' IDENTIFIED BY '1234'
GRANT SELECT ON Contactos TO Manuel
```

        - Podemos indicar `*` en lugar de `NombreTabla` para conceder el permiso en todas las tablas.
        - En el caso del privilegio `UPDATE` se permite especificar qué columnas pueden ser actualizadas.

```
GRANT UPDATE(Nombre, Apellidos) ON Contactos TO Manuel
```

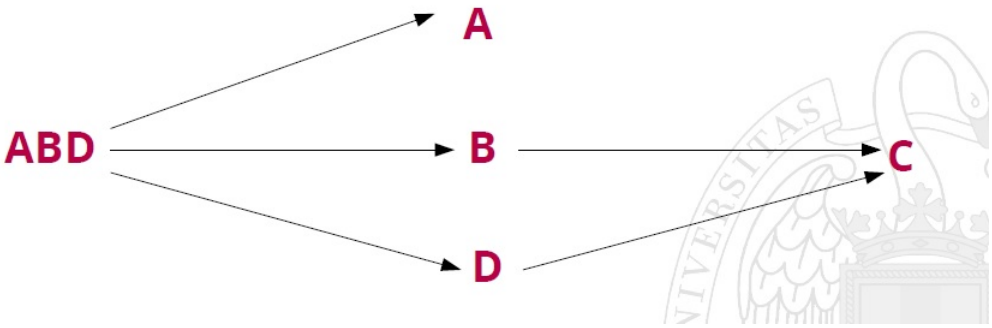

Figura 10: Transferencia de privilegios

- Existe un usuario especial llamado `public`. Los permisos concedidos a dicho usuario se concederán automáticamente a los usuarios nuevos que se añadan a partir de ese momento.

- Revocar permisos:
  - Mediante la cláusula `REVOKE`

```
REVOKE Privilegio1, Privilegio2, ...
ON NombreRelacion
FROM Usuario1, Usuario2, ...
```

  - Ejemplo:

```
REVOKE ALL PRIVILEGES ON * FROM Manuel
```

- Transferencia de privilegios:
  - La cláusula `GRANT` permite conceder al usuario afectado la capacidad de poder conceder el privilegio correspondiente a otros usuarios de la base de datos.
  - Para ello se añade `WITH GRANT OPTION` al final de la cláusula `GRANT`.
  - Ejemplo:

```
GRANT SELECT ON Contactos TO Manuel WITH GRANT OPTION
```

  - El usuario *Manuel* podrá permitir conceder acceso a la tabla *Contactos* a otros usuarios.
  - Supongamos que el usuario A concede un privilegio a B con posibilidad de transferencia, y B concede ese mismo privilegio a C.
  - Si A revoca el privilegio concedido a B, el usuario C también perderá ese mismo privilegio salvo que exista un usuario D que también se lo haya concedido.
  - La concesión de un determinado privilegio a los usuarios se puede representar mediante un grafo dirigido, donde el administrador (ABD) es la raíz (ver figura 10).
  - Un usuario tiene el privilegio correspondiente si existe un camino en el grafo desde el administrador hasta dicho usuario.

3. Introducción a PHP

3.1. Introducción a PHP

3.1.1. ¿Qué es PHP?

- PHP es un lenguaje de scripts que se ejecuta en el lado del servidor.
- Su código está incluido en una página HTML clásica.
- Otros lenguajes de guiones para el servidor: ASP, JSP.
- El resultado de la ejecución del código se integra en la página HTML que es enviada al explorador.
- Este procesamiento del código en el servidor no es visible para el cliente.
- Esta tecnología permite realizar páginas web dinámicas. Su contenido puede ser completado en el momento de la llamada a la página gracias a cierta información. Por ejemplo: la información extraída de un formulario o de una base de datos.

Con el siguiente programa en el servidor:

```
<!DOCTYPE>
<html>
<head>
<title>Mi primera página PHP</title>
</head>
<body>
<p>
<?php echo "Bienvenido al mundo de PHP!"; ?>
</p>
</body>
</html>
```

se construye la siguiente página web:

```
<!DOCTYPE>
<html>
<head>
<title>Mi primera página PHP</title>
</head>
<body>
<p>
Bienvenido al mundo de PHP!
</p>
</body>
</html>
```

3.1.2. Delimitadores PHP

Delimitadores de los scripts de PHP:

- Opción 1:

```
<script language="PHP">
...
</script>
```

- Opción 2:

```
<?php ...?>
```

- También es posible configurar el servidor para admitir otros formatos:

```
<\%...\%>
```

3.1.3. Bloques PHP

Dentro de un bloque PHP (entre dos delimitadores) puede haber:

- Una sola instrucción:

```php
<?php echo "Hola mundo!" ?>
```

- Una secuencia de instrucciones separadas por puntos y comas:

```php
<?php
    $suma = 4;
    $suma += 3;
    echo $suma
?>
```

3.1.4. Comentarios PHP

- Escritura en la página: `echo` y `print`.

```php
<?php
    // De una sola línea
    # Otro de una sola línea
    /* Y este es un comentario de múltiples líneas */
?>
```

3.1.5. Variables y tipos en PHP

- Nombres de variables: `$nombre` (sensible a mayúsculas y minúsculas).
- Lenguaje débilmente tipado:
  - No se declaran las variables (porque no hay que especificar el tipo).
  - Se crean al usarse por primera vez.
  - El tipo se infiere del contexto:

```php
<?php
    $cad = "Hola";
    $bool = true;
    $num = 13;
?>
```

- Conversión automática de tipos en expresiones.
- Tipos básicos:
  - `boolean`
  - `integer`
  - `float`
  - `string`
- Tipos compuestos:
  - `array`
  - `object`
- Tipo especial: `NULL` (constante `NULL`).

3.1.6. Cadenas de caracteres (string)

- Pueden ser de cualquier longitud.
- Cadenas literales:encerradas entre comillas simples ('...') o dobles ("...").
- Para utilizar comillas simples o dobles en una expresión como parte del texto es necesario especificarlas como \' o \":

```
<?php
    echo "Don\'t let me down";
?>
```

- Las cadenas de caracteres de la cadena son accesibles con llaves:

```
<?php
    $cad = "Hola";
    echo $cad{1}; // Muestra la letra 'o';

?>
```

- La primera posición es la 0.

3.1.7. Cadenas de caracteres entre comillas dobles

- Cuando se encuentra un $:
  - Se interpreta lo que sigue como una variable.
  - Se sustituye la variable por su valor.
- Se puede encerrar la variable o su nombre entre llaves:

```
<?php
    $nombre = "Pablo";
    echo "Te llamas $nombre!"; // Te llamas Pablo!
    echo "Te llamas {$nombre}!";
    echo "Te llamas ${nombre}!";
?>
```

3.1.8. Operadores para cadenas

- Concatenación: operador . (punto):

```
<?php
    $nombre = "Pablo";
    $apellido = "Fernández";
    echo $nombre." ".$apellido; // Pablo Fernández
?>
```

- Concatenación y asignación en un paso: operador .=:

```
<?php
    $nombre = "Pablo";
    $nombre .= " Fernández";
    echo $nombre; // Pablo Fernández
?>
```

Algunas funciones útiles para procesar cadenas:

- `strlen()`: devuelve la longitud de una cadena.
- `strpos()`: devuelve la posición donde se encuentre una subcadena (dentro de otra cadena):

```php
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Devuelve un 6.

- `str_replace()`: reemplaza una subcadena por otra:

```php
<?php
echo str_replace("world", "Dolly", "Hello world!");
?>
```

Devuelve `Hello Dolly`.

3.1.9. Arrays

- Los arrays en PHP son mapas con orden, es decir, son pares de parejas clave-valor.
- Claves: pueden ser `integer` o `string`. Pueden estar ambos tipos en un mismo array:

```php
<?php
    $num = 12;
    $matriz = array("foo" => "bar", 12 => true);
    echo $matriz["foo"]; // bar
    echo $matriz[12];    // 1
    }
?>
```

- Si no se indica clave, se toma el máximo entero usado +1:

```php
<?php
    // Este array ...}
    array(5 => 43, 32, 56, "b" => 12);
    // ... es igual que este otro
    array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

- Para PHP las matrices son arrays de arrays.
- Ejemplo:

```php
<?php
    $ciudades = array('ESPAÑA'=> array('Madrid','Barcelona','Zaragoza'),
                      'FRANCIA'=> array('París','Nantes'));
    echo $ciudades['ESPAÑA'][0];
    echo $ciudades['FRANCIA'][1];
?>
```

- Modificación / creación de valores (y eliminación):

```php
<?php
    $matriz = array(5 => 1, 12 => 2);
    $matriz[] = 56; // Igual que $matriz[13] = 56;
    $matriz["x"] = 42; // Nuevo elemento con clave "x"
    unset($matriz[5]); // Elimina ese elemento
    unset($matriz);    // Elimina la matriz completa
?>
```

- Los índices (claves enteras) no se reutilizan.

```php
<?php
    $matriz = array(1, 2, 3); // Índices 0, 1, 2
    unset($matriz[2])  // Eliminamos el valor 3 (índice 2)
    $matriz[] = 4; // $matriz[3] = 4, aunque el 2 esté libre
    $matriz = array_values($matriz); // Reindexa el array
?>
```

3.1.10. Constantes

- Las constantes son sensibles a mayúsculas y minúsculas.
- Definición:

```php
<?php
    define("CONSTANTE", "Hola amigo!");
    echo CONSTANTE; // muestra "Hola amigo!"
?>
```

- El alcance de una costante definida es el script donde se encuentra.
- Existen muchas constantes predefinidas.

3.1.11. Operadores

Son similares a los de C++. Operadores numéricos:

- Operadores aritméticos: `+  - * /  %`
- Operadores de asignación: `= += -= *= /=`
- Operadores de comparación:
- Operadores de incremento/decremento: `++ --`
- Operadores lógicos: `== != < <= >=  ! && || === !==`
  - `===`: igualdad de valores y de tipos. Ejemplo: `100 === '100'` devuelve `false`.
  - `!==`: desigualdad de valores o de tipos. Ejemplo: `100 === '100'` devuelve `true`.

Operadores para cadenas:

- `.` concatenación.
- `.=` concatenación con asignación.

3.1.12. Instrucciones de control

También son similares a las de C++

- `if(condición){...} else {...}` o `if(condicion){...} elseif(condicion) {...}`
- `switch(expresión){`
  `case valor1: ...; break;`
  `case valor2: ...; break;`
  `...`
  `default: ...`
  `}`

Nota: existe una sintaxis ligeramente distinta si queremos intercalar código HTML: `<?php if (condición)?>` `código HTML` `<?php endif; ?>`

- `while(condición){`
  `...`
  `}`
- `do`
  `...`
  `while(condición)`
- `for(inicialización; condición; incremento) {`
  `...`
  `}`

Al igual que en el caso anterior, la sintaxis de los bucles puede variar ligeramente si se intercala código HTML.

3.1.13. Procesamiento de colecciones

Procesamiento de los elementos de un array:

- foreach(array as $valor)...
- foreach(array as $clave => $valor)...
- Se itera para cada elemento del array, asignando el valor a $valor (y la clave a $key en el segundo caso).

```php
<?php
    $matriz = array("a","b","c","d","e");
    foreach($matriz as $val){
        echo "$val";
    };
    foreach($matriz as $key => $val){
        echo "$key: $val";
    }
?>
```

3.1.14. Funciones

- Una función se define a partir de nombre y de sus sus parámetros:
- Sintaxis:

  `function nombre($par1 [=valor1], $par2 [=valor2], ... , $parN [=valorN]) { ...}`
  `...`
  `}`
- Invocación: `nombre($arg1, $arg2, ... , $argN);`
- Se pueden omitir argumentos por el final, usándose entonces los valores por defecto.
- Algunos ejemplos:

```php
<?php
    //Algunas declaraciones:
    function hola(){
       echo "¡Hola!";
    }
    function producto($valor1,$valor2){
        return $valor1*$valor2
    }
    //Algunas llamadas:
```

```
    hola();
    producto(3,7);
?>
```

- Por defecto los parámetros se pasan por valor.
- Paso de parámetro por referencia: `function nombre(&$parámetro) { ... }`
- Devolución de valores: instrucción `return`.
- Ejemplo:

```
<?php
    function foo(&$a, $b="Hola") {}
        ...
    return $res
    ...
    $r = foo($una, $dos);
    $r = foo($una);
?>
```

3.1.15. Ámbito de las variables

- El ámbito de una variable es el script donde se encuentra. Su duración queda restringida al tiempo de duración del script. Si se vuelve a ejecutar el script se vuelve a crear.
- Una variable definida en una función posee un ámbito local (el de la función).
- Una variable definida fuera de una función no es visible desde ella.
- Para saltarse estas limitaciones es posible (aunque no recomendable) usar `global` o `$GLOBALS`.

```
<?php
    $num = 12;
    function foo(){
        global $num; // Acceso a la variable global
        echo $num;
        echo $GLOBALS['num'];
    }
?>
```

3.1.16. Ejemplos de generación de código HTML con PHP

- La idea fundamental de PHP consiste en generar código HTML de una manera dinámica.
- Cómo generar una lista de HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Generacion de listas HTML</title>
</head>
<body>
<?php
__echo "<ul>";
__echo "<li>";
__echo "Primer item";
__echo "</li>";
__echo "<li>";
__echo "Segundo item";
__echo "</li>";
__echo "</ul>";
?>
</body>
</html>
```

- Cómo generar una tabla de HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Tablas con PHP</title>
<style>
__table, th, td {
__  border: 1px solid black;
__}
</style>
</head>
<body>
<?php
__$lista = array('a' =>"avion" ,'b' =>"barco",'c' =>"casa" );
__echo "<table>";
__foreach ($lista as $key => $value) {
____echo "<tr>";
____echo "<td> $key </td>";
____echo "<td> $value </td>";
____echo "</tr>";
__};
__echo "</table>";
?>
</body>
</html>
```

En este ejemplo el papel del array se sustituirá más adelante por el acceso a una base de datos.

3.2. Formularios

3.2.1. Introducción

- En HTML existen dos métodos principales para interactuar con el usuario:
  - Los vínculos.
  - Los formularios.
- Los vínculos simplemente llaman a una página web: interacción sencilla.
- Los formularios son capaces de enviar una gran cantidad de información.
- Los vínculos pueden enviar información a la página que se llama.
  - Sintaxis: `Nombre-url?variable-1=valor-1&...&variable-n=valor-n`.
  - El carácter `?` introduce la lista de parámetros separados por el carácter &. Cada parámetro está constituido por una pareja `variable=valor`.

Nos centraremos en los formularios. Ejemplo:

```
<!DOCTYPE HTML>
<html>
<body>
    <form action="welcome.php" method="post">
        Name: <input type="text" name="name"><br>
        E-mail: <input type="text" name="email"><br>
        <input type="submit">
    </form>
</body>
</html>
```

Para acceder a la información enviada hay que acceder a una variable superglobal (`$_POST`) :

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

En este caso la información enviada está en `$_POST["name"]` y `$_POST[.emᵉail"]`

3.2.2. Acceso a los datos de formularios

- Cuestión: ¿cómo podemos tener acceso a los datos enviados cuando submitimos la información de un formulario.
- Toda la información del formulario se almacena automáticamente en unas variables superglobales: $_POST y $_GET (según sea el valor del atributo `method`: `POST` o `GET`).
- También están disponibles en la variable superglobal `$_REQUEST` que agrupa a las anteriores.

Con la siguiente página web creamos un formulario y le enviamos la información a un script de PHP:

```
<!DOCTYPE html>
<head><title>Entrada de datos</title></head>
  <body>
    <form action="proceso.php" method="post">
    <div>
      Nombre: <input type="text" name="nombre" value="" />
      <input type="submit" name="aceptar" value="Aceptar" />
    </div>
    </form>
  </body>
</html>
```

En este script de PHP accedemos a la información que un formulario le ha enviado:

```
<!DOCTYPE>
  <head>
    <title>Proceso</title>
  </head>
  <body>
    <div>
____<?php
      // Visualización de los datos contenidos
      // en las matrices $_POST y $_REQUEST.
      echo '$_POST[\'nombre\'] -> ',$_POST['nombre'],'<br \>';
      echo '$_REQUEST[\'nombre\'] -> ',$_REQUEST['nombre'],'<br \>';
    ?>
    </div>
  </body>
</html>
```

En este script de PHP accedemos a la información que un formulario le ha enviado:

```
<!DOCTYPE html>
<head><title>Proceso</title></head>
<body>
    <div>
____<?php
      // Visualización de los datos contenidos
      // en las matrices $_POST y $_REQUEST.
      echo '$_POST[\'nombre\'] -> ',$_POST['nombre'],'<br \>';
      echo '$_REQUEST[\'nombre\'] -> ',$_REQUEST['nombre'],'<br \>';
    ?>
    </div>
</body>
</html>
```

- Otra posibilidad para acceder a la información de un formulario consiste en importar en variables PHP la información.
- Función para importar: `import_request_variables(cadena tipos, cadena prefijo)` donde:
  - *tipos*: representa el tipo de información deseada (`P` o `p` para `POST` y `G` o `g` para `GET`).
  - *prefijo*: prefijo para el nombre de la variable.

El último script se puede escribir de la siguiente manera:

```
<!DOCTYPE html>
<head><title>Proceso</title></head>
<body>
    <div>
    __<?php
      // Importando datos del formulario
      //método POST y prefijo form_
      import_request_variables('P','form_');
      echo '$form_nombre = ',$form_nombre,'<br \>';
    ?>
    </div>
</body>
</html>
```

Utilizar una matriz para acceder a los datos enviados

- Es posible utilizar una notación de tipo matriz en el atributo `name` de las etiquetas: `<input>`, `<select>` y `<textarea>`.
- Un ejemplo:

```
<form action="procesar.php" method="POST"><div>
Apellido: <input type="text" name="entrada[]">
Nombre:<input type="text name="entrada[]">
<input type="submit" name="aceptar" value="Aceptar">
</form>
```

- La aparición de `entrada[]` como valor de `name` produce una única variable `entrada` con formato de matriz (array). El primer elemento tiene la clave 0.
- Es posible utilizar una notación de tipo matriz indicando el nombre de clave de `entrada[]`.
- Un ejemplo:

```
<form action="procesar.php" method="POST"><div>
Apellido: <input type="text" name="entrada[apellido]">
Nombre:<input type="text name="entrada[nombre]">
<input type="submit" name="aceptar" value="Aceptar">
</form>
```

Acceso a la información de cada tipo de zona

Acceso a la información de las zonas de texto

- Consideremos que un formulario tiene las siguientes zonas de texto:

```
Apellido:
  <input type="text" name="apellido" value=""
       size="20" maxlength="20" />
  Contraseña:
  <input type="password" name="contraseña" value=""
       size="20" maxlength="20" />
<br />Comentario:<br />
  <textarea name="comentario" rows="4" cols="50"></textarea>
  <br />
```

- En las siguientes variables se encuentra la información: `$_POST[apellido]`, `$_POST[contraseña]` y `$_POST[comentario]`.

Acceso a la información de los botones de opción

- Consideremos que un formulario tiene el siguiente botón de opción:

```
Sexo:
  <input type="radio" name="sexo" value="H" />Hombre
  <input type="radio" name="sexo" value="M" />Mujer
  <input type="radio" name="sexo" value="?"
                      checked="checked" />No sabe
```

- Si se selecciona la opción *Hombre* en la variable `$_POST[sexo]` encontraremos el valor "H".

Acceso a la información de las casillas de verificación

- Consideremos que un formulario tiene el siguientes casillas de verificación:

```
Colores preferidos:
  <input type="checkbox" name="azul" value="b" />Azul
  <input type="checkbox" name="blanco" />Blanco
  <input type="checkbox" name="rojo" />Rojo
  <input type="checkbox" name="nosabe"
                         checked="checked" />No sabe
```

- Si se selecciona la opciones *azul* y *rojo* tendremos: `$_POST[azul]=b` y `$_POST[rojo]=on`.

Acceso a la información de una lista de selección única

- Consideremos que un formulario tiene la siguiente lista de selección única:

```
Idioma:
  <select name="idioma">
    <option value="E" selected="selected">Español</option>
    <option value="F" >Francés</option>
    <option value="I">Italiano</option>
  </select>
```

- Si no se selecciona nada tendremos: `$_POST[idioma]=E`.

Acceso a la información de una lista de selección múltiple

- Consideremos que un formulario tiene la siguiente lista de selección múltiple:

```
<br />Frutas preferidas:<br />
  <select name="frutas[]" multiple="multiple" size="8">
    <option value="A">Albaricoques</option>
    <option value="C">Cerezas</option>
    <option value="F">Fresas</option>
    <option value="P">Melocotones</option>
    <option value="?" selected="selected">
                    No sabe</option>
  </select>
```

- Si se seleccionan *fresas* y *albaricoques* tendremos: `$_POST[frutas]=[A, F]`.

3.2.3. Generación dinámica de formularios

Generación dinámica de formularios Como otros elementos de una página HTML, PHP permite generar dinámicamente los formularios. Algunos casos:

- Generar todo el formulario.
- Generar valores iniciales en entradas de texto.
- Generar una lista de opciones.

Inicialización de un campo de texto En este ejemplo se inicializa una entrada con cierto nombre:

```
<form action="entrada.php" method="POST">
Nombre:<input type="text name="nombre" value="<?php echo $nombre ?>">
<input type="submit" name="aceptar" value="Aceptar">
</form>
```

Se supone que en algún sitio está inicializado $nombre.

Generación de una lista de selección única Ejemplo:

```
<!DOCTYPE html >
<head><title>Generar una lista de opciones de selección única</title></head>
<body>
<div>
__<?php
  // Lista de los idiomas para mostrar en la lista,
  // con la forma de una matriz asociativa que da el código
  // del idioma (clave de la matriz) y el nombre del idioma.
  $idiomas_disponibles = array(
    'E' => 'Español',
    'F' => 'Francés',
    'I' => 'Italiano');
  // Código del idioma del usuario.
  $idioma = 'E';
  ?>
```

Generación de una lista de selección única Ejemplo:

```
      <!-- creación del formulario -->
      <form action="entrada.php" method="POST">
      Idioma:<br />
      <select name="idioma">
      <?php
      // Código PHP que genera la parte dinámica del formulario.
      // Recorrer la lista para mostrar y recuperar el código
      // y el nombre.
      foreach($idiomas_disponibles as $código => $nombre) {
        // Determinar si la línea debe estar seleccionada
        //  - sí, si el código es igual al código del idioma
        //    del usuario
        //  - si es el caso, poner el atributo "selected" en
        //    la etiqueta "option"; si no, no poner nada
        $selección = ($código == $idioma)?'selected':'';
        // Generar la etiqueta "option" con la variable $código
        // la opción "value", la variable $selección
        // para la indicación de selección y la variable $nombre
        // para el texto mostrado en la lista.
        echo "<option value=\"$código\" $selección>$nombre</option>";
      }
      ?>
      </select>
      </form>
    </div>
  </body>
</html>
```

Generación de una lista de selección múltiple Ejemplo:

```
<!DOCTYPE html>
<head><title>Generar una lista de opciones de selección múltiple</title></head>
<body>
<div>
<?php
__// Frutas para mostrar en la lista, con la forma
__// de una matriz asociativa que da el código
__// de la fruta (clave de la matriz) y el nombre de la fruta.
__$frutas_del_mercado = array(
__   'A' => 'Albaricoques',
__   'C' => 'Cerezas',
__   'F' => 'Fresas',
__   'P' => 'Melocotones',
__   '?' => 'No sabe');
__// Frutas preferidas del usuario, con la forma
__// de una matriz que da el código de las frutas correspondientes.
__$frutas_preferidas = array('A','F');
__// Advertencia: veremos más adelante cómo recuperar
__//            esta información en una base de datos.
__?>
```

Generación de una lista de selección múltiple Ejemplo:

```
__<!-- creación del formulario -->
__<form action="entrada.php" method="POST">
__Frutas preferidas:<br />
__<select name="frutas[]" multiple size="8">
__<?php
__// Código PHP que genera la parte dinámica del formulario.
__// Recorrer la lista para mostrar y recuperar el código
__// y el nombre.
__foreach($frutas_del_mercado as $código => $nombre) {
__  // Determinar si la línea debe estar seleccionada
__  //  - sí, si el código figura en la lista de las frutas
__  //    preferidas del usuario => búsqueda de $código
__  //    en $frutas_preferidas con la función in_array
__  //  - si es el caso, poner el atributo "selected" en
__  //    la etiqueta "option"; si no, no poner nada.
__  $selección =
__    in_array($código,$frutas_preferidas)?'selected':'';
__  // Generar la etiqueta "option" con la variable $código
__  // para el atributo "value", la variable $selección
__  // para la indicación de selección y la variable $nombre
__  // para el texto mostrado en la lista.
__  echo "<option value=\"$código\" $selección>$nombre</option>";
__}
__?>
    </select>
__</form>
</body>
</html>
```

3.2.4. Procesamiento de formularios

Existen tres formas para procesar la información de un formulario con scripts de PHP:

- El formulario en HTML puro (sin ningún elemento dinámico) se relaciona con el script de PHP a través del atributo `action`.
- Colocar el formulario en un script de PHP (para construir alguna parte de manera dinámica) y que otro script PHP lo procese.
- Colocar el formulario en un script de PHP que lo construya dinámicamente y lo procese (indicando su propio nombre en la opción `action`).
- Ninguna de estas formas es mejor que las demás. La elección de una u otra depende de cada caso.

3.2.5. Validación de datos

- Limpieza de los espacios innecesarios:
  - Es posible quitar los espacios innecesarios (al principio y al final de la cadena) en las zonas de entrada de texto libre con la función `trim`.
  - Ejemplo: `$nombre=trim($_POST['nombre']);`
- Dato obligatorio:
  - Es muy sencillo comprobar si hay información en una entrada de texto.
  - Ejemplo:

```
$nombre=trim($_POST['nombre']);
if ($nombre==''){
//Procesar respuesta
}
```

Longitud máxima de cadena:

- Con la función `strlen` se puede controlar la longitud de las cadenas de caracteres introducidas en las zonas de entrada de texto.

- Ejemplo:

```
$nombre=trim($_POST['nombre']);
if (strlen($nombre)>25){
//Procesar respuesta
}
```

Cadenas de caracteres con formato:

- Con las funciones `eregi` y `ereg` se puede controlar el formato de una cadena de caracteres.
- `ereg` tiene en cuenta las mayúsculas mientras que `eregi` no.
- Ejemplo:
  - Una cadena de caracteres debe comenzar por una letra, seguida de letras (y caracteres: _,#,*,$). Debe de tener una longitud mínima de 4:

```
$contraseña=trim($_POST['contraseña']);
if (!eregi('^[a-z][a-z0-9_#*$]{3,}',$contraseña)){
//Procesar contraseña inválida
}
```

  - `eregi` no diferencia entre mayúsculas y minúsculas.
  - `^`: marca de comienzo.
  - `[a-z]`: carácter entre *a* y *z*.
  - `[a-z0-9_#*$]{3,}`: le siguen al menos tres caracteres del conjunto de los incluidos.

Validez de una fecha:

- Tomemos, por ejemplo, el formato de fecha: *DD/MM/AAAA*. Por lo tanto, son inválidas: *01/01/99* y *32/01/1999*.
- Comprobación del formato:

```
$fecha_nacimiento=trim($_POST['fecha_nacimiento']);
$formato_fecha= '^[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}$';
if (!ereg($formato_fecha,$fecha_nacimiento)){
//Procesar fecha inválida
}
```

- `^`: marca de comienzo.
- `[0-9]{1,2}`: una o dos cifras.
- `/`: seguido del cáracter `''/''`.
- `[0-9]{4}`:seguido de cuatro cifras.
- `$`: fin de la información de la fecha. No puede haber nada más.

Validez de una fecha:

- Existen tres métodos similares para saber si una fecha con formato correcto es válida.
- Con la función `explode`:

```php
$dma=explode('/',$fecha_nacimiento);
//$dma[0]: el día
//$dma[1]: el mes
//$dma[2]: el año
if (!checkdate($dma[1],$dma[0],$dma[2])){
//Procesar fecha inválida
}
```

- Con la función `explode` y `list`:

```php
$list($dia,$mes,$año)=explode('/',$fecha_nacimiento);
if (!checkdate($dia,$mes,$año)){
//Procesar fecha inválida
}
```

Validez de una fecha:

- Con la función `ereg`:

```php
$formato_fecha= '^[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}$';
if (!ereg($formato_fecha,$fecha_nacimiento,$dma)){
//Procesar fecha con formato incorrecto
} else {
//dma[1]: el día
//dma[2]: el mes
//dma[3]: el año
if (!checkdate($dma[2],$dma[1],$dma[3])){
//Procesar fecha inválida
}
```

3.2.6. Juntándolo todo: un ejemplo

Un ejemplo Deseamos validar el siguiente formulario

```php
<form method="post" action="index2.php">
    <label> Nombre </label>
    <br />
    <input type="text" name="nombre" value="<?php echo $nombre ?>" />
    <br />
    <label> Edad </label>
    <br />
    <input type="text" name="edad" size="3" value="<?php echo $edad ?>" />
    <br />
    <label> E-mail </label>
    <br />
    <input type="text" name="email" value="<?php echo $email ?>" />
    <br />
    <input type="submit" value="Enviar" />
</form>
```

con las siguientes restricciones:

- El nombre es un campo obligatorio.
- La edad tiene que estar entre 3 y 130 años.
- La dirección de correo electronica tiene que ser correcta.

Declaramos un página validaciones.php auxiliar:

```php
<?php
 function validaRequerido($valor){
    if(trim($valor) == ''){
       return false;
    }else{
       return true;
    }
 }
 function validarEntero($valor, $opciones=null){
    if(filter_var($valor, FILTER_VALIDATE_INT, $opciones) === FALSE){
       return false;
```

```
    }else{
        return true;
    }
}
function validaEmail($valor){
    if(filter_var($valor, FILTER_VALIDATE_EMAIL) === FALSE){
        return false;
    }else{
        return true;
    }
}
?>
```

Comentarios:

- `filter_var($valor, FILTER_VALIDATE_INT, $opciones)` es un filtro que permite validar si una variable contiene un valor numérico. Opcionalmente (como en este caso) se puede indicar unos valores máximos y mínimos permitidos.
- La función `validarEntero` funcionaría también si no pasamos un intervalo de máximos y mínimos ya que `$opciones` tiene un valor por defecto.
- `filter_var($valor, FILTER_VALIDATE_EMAIL)`: es un filtro (predefinido) que permite validar si una variable contiene una dirección de correo electrónico válida.

Declaramos un otra página validado.php auxiliar para indicar que todo ha ido bien:

```
<!DOCTYPE>
<html>
    <head>
        <title> Formulario </title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    </head>
    <body>
        <strong> Sus datos han sido enviados correctamente </strong>
    </body>
</html>
```

La función `isset` determina si una variable está definida (tiene contenido) o no. Juntándolo todo:

```
<?php
//Importamos el archivo con las validaciones.
require_once 'validaciones.php';
//Guarda los valores de los campos en variables, siempre y cuando se haya enviado el formulario, sino se guardará
     null.
$nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
$edad = isset($_POST['edad']) ? $_POST['edad'] : null;
$email = isset($_POST['email']) ? $_POST['email'] : null;
//Este array guardará los errores de validación que surjan.
$errores = array();
//Pregunta si está llegando una petición por POST, lo que significa que el usuario envió el formulario.
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    //Valida que el campo nombre no esté vacío.
    if (!validaRequerido($nombre)) {
        $errores[] = 'El campo nombre es incorrecto.';
    }
    //Valida la edad con un rango de 3 a 130 años.
    $opciones_edad = array(
        'options' => array(
            //Definimos el rango de edad entre 3 a 130.
            'min_range' => 3,
            'max_range' => 130
        )
    );
    if (!validarEntero($edad, $opciones_edad)) {
        $errores[] = 'El campo edad es incorrecto.';
    }
```

```
    //Valida que el campo email sea correcto.
    if (!validaEmail($email)) {
        $errores[] = 'El campo email es incorrecto.';
    }
    //Verifica si ha encontrado errores y de no haber redirige a la página con
    //el mensaje de que pasó la validación.
    if(!$errores){
        header('Location: validado.php');
        exit;
    }
}
?>
```

```
<!DOCTYPE>
<html>
    <head>
        <title> Formulario </title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    </head>
    <body>
        <?php if ($errores): ?>
            <ul>
                <?php foreach ($errores as $error): ?>
                    <li> <?php echo $error ?> </li>
                <?php endforeach; ?>
            </ul>
        <?php endif; ?>
        <form method="post" action="index2.php">
            <label> Nombre </label>
            <br />
            <input type="text" name="nombre" value="<?php echo $nombre ?>" />
            <br />
            <label> Edad </label>
            <br />
            <input type="text" name="edad" size="3" value="<?php echo $edad ?>" />
            <br />
            <label> E-mail </label>
            <br />
            <input type="text" name="email" value="<?php echo $email ?>" />
            <br />
            <input type="submit" value="Enviar" />
        </form>
    </body>
</html>
```

Comentarios:

- `require_once 'validaciones.php';` permite incluir las funciones de validación.
- Con una lista no ordenada (generada dinámicamente) se muestran los errores contenidos en el array `$errores`.
- Con `header('Location: validado.php');` se muestra la página una página que indica que no ha habido errores.

4. Bases de datos con PHP

4.1. Introducción

- Cuestión: persistencia de la información.
- La utilización de bases de datos es el método estándar para el almacenamiento de datos en la web. Ejemplos:
  - Catálogos de productos.
  - Lista de clientes.
  - Lista de transacciones realizadas, etc.
- PHP permite una conexión a una gran número de sistemas de base de datos: MySQL, ORACLE, Microsoft SQL Server, Informix, SQLite, etc.
- PHP admite ODBC (*Open DataBase Connectivity*) y, por lo tanto, puede conectarse a cualquier base de datos que soporte ODBC.
- La versión 5 de PHP ofrece dos extensiones para trabajar con BB.DD.:
  - MySQL (prefijo `mysql_`).
  - MySQLi (prefijo `mysqli_`).
- Ambas extensiones son muy parecidas.

Operaciones con las bases de datos en MySQL:

- Conexión/desconexión.
- Selección de la base de datos.
- Consultas.
- Actualizaciones.

4.2. Conexión

`mysqli_connect`: permite establecer una conexión con una base de datos MySQL. Parámetros:

- `host`: nombre o dirección IP al debe debe conectarse. Ejemplo: *localhost*.
- `usuario`: nombre del usuario debe de utilizarse para establecer la conexión. Ejemplo: *root@localhost*.
- `contraseña`: cadena vacía, significa sin contraseña.
- `nombre_base`: nombre de la base de datos (parámetro opcional).
- `puerto`: número de puerto para la conexión al servidor MySQL (parámetro opcional).

La función `mysqli_connect` devuelve un objeto con un identificador `mysqli` o el valor `FALSE` en caso de error. `mysqli_close`: permite cerrar una conexión durante la ejecución de un script. Parámetros:

- `conexión`: identificador de conexión devuelto por `mysqli_connect`.

La función `mysqli_close` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error. Ejemplo:

```
<title>Conexión y desconexión</title>
<?php
    // Definición de una pequeña función que abre una conexión.
    function conectar($host,$usuario,$contrasenia='') {
      $db = @mysqli_connect($host,$usuario,$contrasenia);
      if ($db) {
        echo 'Conexión realizada correctamente.<br />';
        echo 'Información sobre el servidor: ',
             mysqli_get_host_info($db),'<br />';
        echo 'Versión del servidor: ',
             mysqli_get_server_info($db),'<br />';
      } else {
        printf(
          'Error %d: %s.<br />',
          mysqli_connect_errno(),mysqli_connect_error());
      }
      return $db;
    }
    // Definición de una pequeña función que cierra una conexión.
    function desconectar($conexion) {
      if ($conexion) {
        $ok = @mysqli_close($conexion);
        if ($ok) {
          echo 'Desconexión realizada correctamente.<br />';
        } else {
          echo 'Fallo en la desconexión. <br />';
        }
      } else {
          echo 'Conexión no abierta.<br />';
      }
}
```

```
// Primera prueba de conexión/desconexión.
echo '<b>Primera prueba</b><br />';
$db = conectar('localhost','blas','web');
desconectar($db);
// Segunda prueba de conexión/desconexión.
echo '<b>Segunda prueba</b><br />';
$db = conectar('xampp','desconocido','desconocido');
desconectar($db);
?>
```

Comentarios:

- Utilizamos el carácter `@` para no mostrar las alertas generadas por las funciones en caso de error.
- `mysqli_connect_errno()`: devuelve el número que corresponde a cada error (un cero si no hay error).
- `mysqli_connect_error()`: devuelve una cadena de caracteres con la descripción del error (la cadena vacía si no hay error).

4.3. Selección de una base de datos

- `mysqli_select_db` permite seleccionar (si no se ha hecho con `mysqli_connect`) o modificar la base de datos seleccionada.
- Parámetros:
  - `conexión`: identificador devuelto por `mysqli_connect`.
  - `nombre_base`: nombre de la base de datos.
- La función `mysqli_select_db` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error.
- Con PHP también podemos crear una base de datos. Ejemplo:

```php
<?php
    $db = @mysqli_connect('localhost','root','');
    if ($db) {
        echo 'Conexión realizada correctamente.<br />';
        echo 'Información sobre el servidor: ',
             mysqli_get_host_info($db),'<br />';
        echo 'Versión del servidor: ',
             mysqli_get_server_info($db),'<br />';
    } else {
        printf(
          'Error %d: %s.<br />',
          mysqli_connect_errno(),mysqli_connect_error());
    };
    // Create database
    $sql = "CREATE DATABASE myDB";
    if (mysqli_query($db, $sql)) {
        echo "Database created successfully";
    } else {
        echo "Error creating database: " . mysqli_error($db);
    }
    @mysqli_close($db);
 ?>
```

Y crear tablas:

```php
<?php
    $db = @mysqli_connect('localhost','root','','mydb');
    if ($db) {
        echo 'Conexión realizada correctamente.<br />';
        echo 'Información sobre el servidor: ',
             mysqli_get_host_info($db),'<br />';
        echo 'Versión del servidor: ',
             mysqli_get_server_info($db),'<br />';
    } else {
        printf(
          'Error %d: %s.<br />',
          mysqli_connect_errno(),mysqli_connect_error());
    };
    // sql to create table
    $sql = "CREATE TABLE categorias (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    categoria VARCHAR(30) NOT NULL
    )";
    if (mysqli_query($db, $sql)) {
        echo "Table categorias created successfully";
    } else {
        echo "Error creating table: " . mysqli_error($db);
    }
    @mysqli_close($db);
?>
```

4.4. Consultas

`mysqli_query` permite realizar una consulta en una base de datos. Parámetros:

- `conexión`: identificador de conexión devuelto por `mysqli_connect`.
- `consulta`: cadena de caracteres que contiene la consulta que se va a ejecutar.

`mysqli_query` devuelve un identificador (de tipo `mysqli_result`) en caso de éxito y el valor `FALSE` en caso contrario.

Con la función `mysqli_num_rows` podemos conocer el número de filas del resultado.

- La ejecución de una consulta devuelve un resultado que puede ser leído por las siguientes funciones:
  - `mysqli_fetch_array`
  - `mysqli_fetch_assoc`
  - `mysqli_fetch_object`
  - `mysqli_fetch_row`
- Básicamente, estas funciones hacen lo mismo: leen una fila del resultado y avanzan el puntero a la siguiente fila.
- Si no hay ninguna fila que leer devuelven el valor `NULL`.

Diferencia de formatos entre estas funciones:

- `mysqli_fetch_assoc`: devuelve una matriz asociativa cuya clave es el nombre de la columna.
- `mysqli_fetch_row`: devuelve una matriz con enteros como índices.
- `mysqli_fetch_object`: devuelve la fila actual con formato de objeto.
- `mysqli_fetch_array`: depende de un parámetro opcional:
  - `MYSQLI_NUM`: matriz de índices enteros como `mysqli_fetch_row`.
  - `MYSQLI_ASSOC`: matriz asociativa como `mysqli_fetch_assoc`.
  - `MYSQLI_BOTH`: ambos a la vez. Es la opción predeterminada.

Sintaxis:

matriz `mysqli_fetch_array`(*objeto* resultado, tipo)
matriz `mysqli_fetch_assoc`(*objeto* resultado)
objeto `mysqli_fetch_object`(*objeto* resultado)
matriz `mysqli_fetch_row`(*objeto* resultado)

donde:

- *resultado*: es el identificador devuelto por `mysqli_query`.
- *tipo*: algunas de las opciones de `mysqli_fetch_array`

Ejemplos de fetch utilizando las funciones anteriores:

```
<!DOCTYPE html>
<html>
<body>

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    while($row = mysqli_fetch_assoc($result)) {
        echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]." ".$row["lastname"]."</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}

mysqli_close($conn);
?>

</body>
</html>
```

4.5. Actualizaciones

- Actualizar datos supone ejecutar consultas `INSERT`, `UPDATE` o `DELETE`.
- La función `mysqli_affected_rows` permite conocer el número de filas afectadas por la última consulta `INSERT`, `UPDATE` o `DELETE`.
- La función `mysqli_insert_id` devuelve el último valor generado para la columna con el tipo `AUTO_INCREMENT` por una consulta `INSERT`

Actualización:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql)) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Inserción:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
```

```php
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Borrado:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if (mysqli_query($conn, $sql)) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

4.6. Consultas preparadas

- Una consulta preparada es una consulta parametrizada.
- Los parámetros se representan por el carácter: "?".
- En cada ejecución de este tipo de consultas se utiliza el valor actual de las variables PHP asociadas a los parámetros.
- Su interés consiste en poder utilizar varias veces la misma consulta con valores diferentes sin tener que analizar de nuevo la consulta. Mejora el rendimiento.

Pasos a realizar para realizar una consulta preparada:

- Preparar la consulta (`mysqli_prepare`).
- Vincular las variables PHP con los parámetros de la consulta (`mysqli_stmt_bind_param`).
- Ejecutar la consulta (`mysqli_stmt_execute`).
- Conocer el número de filas del resultado (`mysqli_num_rows`).
- Vincular las variables PHP a las columnas del resultado (`mysqli_bind_result`).
- Extraer las filas del resultado (`mysqli_stmt_fetch`).
- Cerrar la consulta preparada (`mysqli_stmt_close`).

Sintaxis de `mysqli_stmt_bind_param`:

- `booleano mysqli_stmt_bind_param(objeto consulta, cadena tipos, mixto variables)`.
- `consulta` es el nombre de la consulta que hemos parametrizado.
- `tipos` es una cadena de caracteres donde indicamos los tipos de las variables a vincular. Casos:
  - `i`: variable de tipo entero.
  - `d`: variable de tipo decimal.
  - `s`: variable de tipo cadena de caracteres.
- `variables`: lista de variables a vincular.

Ejemplo:

```
<!DOCTYPE html>
  <head><title>Consulta preparada: lectura</title></head>
  <body>
—<?php
        // Conexión y selección de la base de datos.
        $db = mysqli_connect('localhost','eniweb','web','eni');
        if (! $db) {
          exit('Fallo en la conexión.');
        }
        // Preparación de la consulta.
        $sql = 'SELECT id,titulo FROM libro WHERE id_coleccion = ?';
        $consulta = mysqli_prepare($db, $sql);
        // Enlace de los parámetros.
        $ok = mysqli_stmt_bind_param($consulta,'i',$id_coleccion);
        // Ejecución de la consulta.
        $id_coleccion = 1;
        $ok = mysqli_stmt_execute($consulta);
        // Enlace de las columnas del resultado.
        $ok = mysqli_stmt_bind_result($consulta,$id,$título);
        // Lectura del resultado.
        echo "<b>Colección número $id_coleccion</b><br />";
        while (mysqli_stmt_fetch($consulta)) {
          echo "$id - $título<br />";
        }
```

```
        // Nueva ejecución y lectura del resultado
        // (no vale rehacer los enlaces).
        $id_coleccion = 3;
        $ok = mysqli_stmt_execute($consulta);
        echo "<b>Colección número $id_coleccion</b><br />";
        while (mysqli_stmt_fetch($consulta)) {
          echo "$id - $título<br />";
        }
        // Desconexión.
        $ok = mysqli_close($db);
    ?>
  </body>
</html>
```

4.7. Actualizaciones preparadas

Pasos a realizar para realizar una consulta preparada:

- Preparar la consulta (`mysqli_prepare`).
- Vincular las variables PHP con los parámetros de la consulta (`mysqli_stmt_bind_param`).
- Ejecutar la consulta (`mysqli_stmt_execute`).
- Conocer el número de filas del resultado (`mysqli_affected_rows`).
- Conocer el valor del último identificador generado para una columna con el tipo `AUTO_INCREMENT` (`mysqli_stmt_insert_id`).
- Cerrar la consulta preparada (`mysqli_stmt_close`).

Ejemplo:

```
<!DOCTYPE html>
<head><title>Consulta preparada: actualización</title></head>
<body>
<?php
__// Conexión y selección de la base de datos.
__$db = mysqli_connect('localhost','eniweb','web','eni');
__if (! $db) {
____exit('Fallo en la conexión.');
__}
__// Preparación de la consulta.
__$sql = 'UPDATE coleccion SET gastos_siniva = ? ' .
________ 'WHERE gastos_siniva IS NULL';
__$consulta = mysqli_prepare($db, $sql);
__// Enlace de los parámetros.
__$ok = mysqli_stmt_bind_param($consulta,'d',$gastos_siniva);
```

```
__// Ejecución de la consulta.
__$gastos_siniva = 1;
__$ok = mysqli_stmt_execute($consulta);
__echo 'Número de colecciones modificadas = ',
__mysqli_stmt_affected_rows ($consulta),'<br />';
__// Preparación de la consulta.
__$sql = 'INSERT INTO coleccion(nombre) VALUES(?)';
__$consulta = mysqli_prepare($db, $sql);
__// Enlace de los parámetros.
__$ok = mysqli_stmt_bind_param($consulta,'s',$nombre);
__// Ejecución de la consulta.
__$nombre = 'Soluciones Informáticas';
__$ok = mysqli_stmt_execute($consulta);
__echo 'Identificador de la nueva colección = ',
_____ mysqli_stmt_insert_id($consulta),'<br />';
__// Desconexión.
__$ok = mysqli_close($db);
 ?>
</body>
</html>
```

5. Sesiones

Introducción:

- El protocolo HTTP es un protocolo "sin estado":
  - Nada permite saber en qué página ha estado antes el usuario.
  - Nada permite saber qué usuario solicita la página.
- Para un sitio web interactivo es necesario identificar a un usuario que recorre las páginas de su web (no pensar que son dos usuarios distintos) y, en general, conservar cierta información del usuario de una página a otra.
- El término *sesión* designa el periodo de tiempo correspondiente a la navegación continua de un usuario en un sitio web.
- *Gestionar las sesiones* significa estar en condiciones de identificar el momento en que un nuevo usuario accede a la página y conservar información hasta que abandona el sitio web.

PHP ofrece un conjunto de funciones que facilitan la gestión de las sesiones de acuerdo con los siguientes principios:

- Un identificador único es automáticamente atribuido a cada sesión.
- Este identificador es transmitido de una página a otra.
- Los datos que se desean conservar durante la sesión se indican a PHP.

Principales funciones:

- `session_start`: abre una nueva sesión o reactiva la sesión actual.
- `session_id`: devuelve (o modifica) el identificador de la sesión.
- `session_name`: devuelve (o modifica) la variable en la que se almacena el identificador de la sesión.
- `session_destroy`: elimina la sesión.

La matriz `$_SESSION` permite manipular fácilmente las variables de la sesión. `session_start`:

- Esta función consulta el entorno para detectar si una sesión ha sido abierta para el usuario actual. Si es así, los datos guardados son recuperados. En caso contrario, se abre una nueva sesión con la atribución de un identificador.
- Esta función devuelve siempre `TRUE`.
- Cualquier script relacionado con la gestión de sesiones tiene que invocar esta función para tener acceso a las variables de sesión.
- Si no se ha abierto todavía la sesión, está función intentará almacenar una *cookie* que contenga el identificador de sesión en el sistema del usuario.
- Después de llamar a la función `session_start`, los datos de la sesión puede manipularse directamente en la matriz `$_SESSION`.
- Para leer o modificar un valor de esta matriz solamente hay que acceder a él utilizando su clave.
- Para guardar un nuevo dato en la sesión, solo hay que almacenar este dato en esta matriz con la clave que se quiera.

`session_id`:

- Invocada sin parámetro devuelve el valor del identificador de la sesión.
- Si no se ha invocado antes `session_start` no tendrá ningún valor.
- Si se invoca con un parámetro modifica el identificador de la sesión. Utilidad de poco interés en la mayoría de los casos.

`session_name`:

- Invocada sin parámetro devuelve el nombre de la variable en la que se almacena el identificador de la sesión.
- Siempre devuelve un valor aunque no se haya llamado a la función `session_start`.
- Si se invoca con un parámetro modifica el nombre de la variable. Utilidad de poco interés en la mayoría de los casos.

`session_destroy`:

- Después de llamar a esta función la sesión deja de existir. Una llamada posterior a `session_start` abrirá una nueva sesión.
- Esta función no elimina los datos de la sesión hasta que no finalice la ejecución del script actual. Para eliminar inmediatamente la información se puede asignar una matriz vacía a `$_SESSION`.
- Tampoco destruye la *cookie* de sesión utilizada.

pagina-1.php

```
<?php
// Abrir/reactivar la sesión.
session_start();
// Guardar dos datos en la sesión.
$_SESSION['nombre'] = 'Olivier';
$_SESSION['datos'] =  // es una matriz...
      array('nombre'=>'Olivier','apellido'=>'Heurtel');
?>
<!DOCTYPE html>
  <head><title>Sesión - Página 1</title></head>
  <body>
    <div><a href="pagina-2.php">Página 2</a></div>
  </body>
</html>
```

pagina-2.php

```
<?php
// Llamada a session_start.
session_start();
?>
<!DOCTYPE html>
  <head><title>Sesión - Página 2</title></head>
  <body>
    <div>
<?php
// Llamada a session_start.
session_start();
// Visualización.
echo '$_SESSION[\'nombre\'] = ',
      isset($_SESSION['nombre'])?$_SESSION['nombre']:'',
      '<br />';
echo '$_SESSION[\'datos\'][\'apellido\'] = ',
      isset($_SESSION['datos']['apellido'])?
                  $_SESSION['datos']['apellido']:'',
      '<br />';
?>
    </div>
  </body>
</html>
```