La Sénia
Institut d'Educació Secundària
Paiporta

UP5. Utilización de Técnicas de Acceso a Datos

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2025-2026

Contenidos

- Utilización de bases de datos relacionales.
- API de PHP para MySQL.
- Establecimiento de conexiones.
- Ejecución de sentencias SQL (DML).
- Consultas preparadas.
- Obtención de resultados.
- Uso de conjuntos de resultados.
- Otros orígenes de datos. Ficheros. XML. JSON
- Buenas prácticas

Utilización de BD relacionales

PHP tiene un soporte muy amplio para BD. Puede interactuar con cualquier base de datos, ya sea relacional o no. Incluso soporta ODBC que le permite acceder a BD de las que no tiene extensiones.

Las extensiones disponibles para MySQL son las siguientes:

- PHP Data Objects (PDO): Cada BD requiere el uso de un controlador de interfaz PDO específico para dicha BD. Por ejemplo, para MySQL es **PDO_MYSQL**.
- Extensión de BD específica del proveedor: que en el caso de MySQL se debe usar **mysqli**.

Como recordatorio, las DBR se estructuran en tablas relacionadas, cada una de ellas representada por tuplas o filas con los datos y columnas, campos o atributos que dan nombre al dato almacenado en la celda

Ver: https://www.php.net/manual/es/refs.database.php

Comparación API de MySQL (mysqli vs pdo)

Para probar la conexión devolveremos el primer DNI de la tabla clientes, el 00371569G

```
<?php
// conexión usando mysqli
try {
    $mysqli = new mysqli("localhost:33006",USERNAME, PASSWORD,"EMPRESA");
    $resultado = $mysqli->query("SELECT DNI AS ID_CLIENTE FROM CLIENTE");
    $fila = $resultado->fetch_assoc();
    echo "El ID de Cliente (mysqli) es " . htmlentities($fila['ID_CLIENTE'] . "<br>\n");
} catch (mysqli_sql_exception $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$mysqli->close();
```

El ID de Cliente (mysqli) es 00371569G
El ID de Cliente (pdo) es 00371569G

```
// conexión usando PDO
try {
    $pdo = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME, PASSWORD);
    $resultado = $pdo->query("SELECT DNI AS ID_CLIENTE FROM CLIENTE");
    $fila = $resultado->fetch(PDO::FETCH_ASSOC);
    echo "El ID de Cliente (pdo) es " . htmlentities($fila['ID_CLIENTE'] . "<br>\n");
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$pdo = null;
?>
```


Ver: https://www.php.net/manual/es/mysqlinfo.api.choosing.php

Elección API de MySQL

- La mayoría desarrolladores se decantan por la API **PDO** ya que la razón principal que aducen es que, gracias a los drivers que soporta, permite acceder a diversas bases de datos (12 en concreto) en lugar de tener que usar las API específicas de los proveedores lo que simplificaría un hipotético cambio de proveedor de la BD (Por ejemplo migrar de Oracle a MySQL)
- Ambas ofrecen una API orientada a objetos aunque msqli tiene interfaz procedimental que puede usarse con programación estructurada
- PDO ofrece mayor seguridad ante ataques de inyección de SQL ya que permite ejecutar sentencias preparadas en el cliente de modo que el cliente sólo introduce los parámetros y no crea la sentencia. Ver: https://www.php.net/manual/es/pdo.prepared-statements.php
- En este enlace podéis ver una comparativa de varios casos de uso de PDO y mysqli muy interesante: https://websitebeaver.com/php-pdo-vs-mysqli

En este tema se usará PDO_MYSQL para los ejemplos y ejercicios

Ver clase PDO: https://www.php.net/manual/es/class.pdo.php

Establecimiento de conexiones

Las conexiones se establecen creando una instancia de la clase PDO independientemente del driver que deseemos usar. Debemos indicar el nombre del origen de datos o DSN (DataBase Source Name) que es como se muestra:

mysql:host=198.0.4.221**;port**=3306**;dbname**=pruebaBD

//

mysql:host=127.0.0.1**:**3306**;dbname**=pruebaBD

//

mysql:host=bd.ejemplo.com**;dbname**=pruebaBD

Ver: https://www.php.net/manual/es/ref.pdo-mysql.connection.php

Además del DSN, podemos pasar el usuario y la contraseña de forma opcional y capturar cualquier error en la conexión:

```
<?php
try{
    $mbd = new PDO('mysql:host=localhost;dbname=pruebaBD', 'usuario', 'contraseña');
} catch (PDOException $e) {   //Ver: https://www.php.net/manual/en/class.pdoexception.php
        print "Error al conectar con la BD: " .  $e->getMessage();
}
?>
```


Ver: https://www.php.net/manual/es/pdo.connections.php

Atributos de la conexión

Las conexiones permiten establecer atributos de la misma con el método de PDO **setAttribute**. Por ejemplo, veamos las alternativas en cuanto a los valores de ***informe de Errores*** de **PDO::ATTR_ERRMODE**

- Modo **Silencioso** (por defecto): Si el método ejecutado en try falla en su ejecución, devuelve el valor false. Se debe comprobar el valor devuelto por el método y usar el método errorInfo() de PDO para obtener detalles del problema. El valor es **PDO::ERRMODE_SILENT**
- Mostrar **Advertencias**: En este modo las funciones se comportan como en el modo silencio (sin lanzar excepciones y devolviendo false al producirse un error) pero el motor PHP genera un mensaje de advertencia o warning. Dependiendo de cómo está configurado el manejo de errores, este mensaje puede mostrarse por pantalla o en un fichero de log. El valor es **PDO::ERRMODE_WARNING**
- Mostrar **Excepciones**: En este modo, se lanzan excepciones para que puedan ser gestionadas antes de mostrar un error o bien se detalle información del mismo. El valor es **PDO::ERRMODE_EXCEPTION**

Ver: https://www.php.net/manual/en/pdo.setattribute.php

Atributos de la conexión – Modo Silencioso

```
// El constructor siempre lanza una excepción si falla
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: "  . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* El primer elemento de $error es el código de error SQLSTATE (standard en las
BD). El segundo elemento es el código específico del motor de BD (mysql). El tercer
elemento es el mensaje textual describiendo el error concreto */
}
```

/* resultado */

```
No se ha podido realizar la inserción!
SQL Error=42S22, DB Error=1054, Message=Unknown column 'TIPO' in 'field list'
```

Atributos de la conexión – Modo Advertencia

```
try {// El constructor siempre lanza una excepción si falla
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* El primer elemento de $error es el código de error SQLSTATE (standard en las
BD). El segundo elemento es el código específico del motor de BD (mysql). El tercer
elemento es el mensaje textual describiendo el error concreto */
}
```

/* resultado */
PHP Warning: PDO::exec(): SQLSTATE[42S22]: Column not found: 1054 Unknown column 'TIPO' in 'field list' in C:\Users\Silvia\DWES\UP5_ADD\atributosConexion.php on line 30
No se ha podido realizar la inserción!
SQL Error=42S22, DB Error=1054, Message=Unknown column 'TIPO' in 'field list'

Atributos de la conexión – Modo Excepción

```
try { // El constructor siempre lanza una excepción si falla
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* $error[0]=SQLSTATE,  $error[1]=código mysql, $error[2]=error concreto */
}
```

/* resultado */

PHP Fatal error: Uncaught PDOException: SQLSTATE[42S22]: Column not found: 1054 Unknown column 'TIPO' in 'field list' in C:\Users\Silvia\DWES\UP5_ADD\atributosConexion.php:46
Stack trace:
#0 C:\Users\Silvia\Documents\2DWES\DWES\UP5_ADD\EjemplosUD9_AD\atributosConexion.php(46): PDO->exec()
#1 {main}
  thrown in C:\Users\Silvia\DWES\UP5_ADD\atributosConexion.php on line 46 **(Puede no mostrarse por pantalla dependiendo de php.ini)**

Nota: La visualización de Excepciones y Warnings depende de la configuración de error_reporting. Con Debug, sí podéis comprobar estos errores en cualquier caso

Ver configuración informe errores: https://www.php.net/manual/en/function.error-reporting

Ejecución de sentencias SQL

- Como se ha introducido en los ejemplos anteriores, para poder ejecutar las sentencias SQL (normalmente DML: INSERT, UPDATE, DELETE y SELECT), hacemos uso del método **exec()** de la instancia PDO.
- Dicho método exec() ejecuta la sentencia SQL ***devolviendo el número de filas afectadas por la sentencia***, no devuelve el resultado de la sentencia SQL ejecutada. Existe el método **query()** que sí devuelve el resultado de la sentencia SQL (ver https://www.php.net/manual/es/pdo.query.php)
- Las sentencias DDL y DCL no deben poder ser ejecutadas por los desarrolladores, deben ser ejecutadas (siempre de forma local en la máquina en la que reside el servidor de la BD) por los Administradores de la BD (DBA).

A continuación se verán ejemplos de ejecución de estas sentencias DML de SQL.

Ver: https://www.php.net/manual/es/pdo.exec

INSERT

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
//Ejemplo INSERT (Se deben haber insertado proveedores con anterioridad)
$affectedRows = $db->exec("INSERT INTO PRODUCTO (COD_PROD, NOMBRE,
PROVEEDOR,PVP)
        VALUES ('P0001', 'MONITOR','A12345678', 200.50),
        ('P0002', 'TECLADO','A12345678',25.49),
        ('P0003', 'RATÓN','A12345678', 15)");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han insertado " . $affectedRows . " filas.\n";
}
```

/* resultado */
Se han insertado 3 filas.

Ver: https://www.php.net/manual/en/pdo.exec

UPDATE

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}

//Ejemplo UPDATE Incrementar 10% el PVP de los productos que sea inferior a 50,5€
$affectedRows = $db->exec("UPDATE PRODUCTO SET PVP=(PVP*1.10)
      WHERE PVP < 50.50;");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la actualización!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han actualizado " . $affectedRows . " filas.";
}
```

/* resultado */
Se han actualizado 2 filas.

DELETE

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}

//Ejemplo DELETE: Eliminar los productos con precio mayor de 200€
$affectedRows = $db->exec("DELETE FROM PRODUCTO WHERE PVP>200;");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar el borrado!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han eliminado " . $affectedRows . " filas.";
}
```

/* resultado */
Se han eliminado 1 filas.

SELECT

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME, PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
//Ejemplo SELECT Obtener los productos con PVP menor de 50€
$result = $db->query("SELECT * FROM PRODUCTO WHERE PVP<50;");
//necesitamos saber el número de filas afectadas en $numRows
$numRows = $db->query("SELECT COUNT(*) FROM PRODUCTO WHERE PVP<50;")-
>fetchColumn();
if ($numRows) {
    print "Se han obtenido " . $numRows . " filas.\n";
    print "El resultado de la consulta es: \n";
    printf("%-10s%-15s%-12s%-7s\n", "CODIGO", "NOMBRE", "PROVEEDOR", "PVP");
    printf("%-10s%-15s%-12s%-7s\n", "------", "------", "---------", "-----");
    foreach ($result->fetchAll() as $row) {
        printf("%-10s%-15s%-12s%-7s\n", $row['COD_PROD'], $row['NOMBRE'] ,
            $row['PROVEEDOR'], $row['PVP']);
    }
} else {
    print "No se han obtenido resultados a mostar!\n";
}
```

/* resultado */

Se han obtenido 2 filas.
El resultado de la consulta es:

CODIGO	NOMBRE	PROVEEDOR	PVP
P0002	TECLADO	A12345678	28.04
P0003	RATÓN	A12345678	16.50

Ver: https://www.php.net/manual/es/pdostatement.fetchcolumn

Consultas preparadas

Las consultas preparadas pueden ser ejecutadas múltiples veces con parámetros distintos y previenen la inyección de SQL en los formularios de forma que un usuario malintencionada pueda cambiar la estructura de tablas, eliminar datos, borrar tablas, etc. Para ello se construye la consulta parametrizando los atributos a tratar de modo que no se pueda ejecutar una sentencia SQL en un atributo ya que añade comillas automáticamente a los parámetros. Dicha sentencia, se crea con el método **prepare()** de PDO indicando con **?** o **:atributo** los atributos cuyo valor espera y se ejecuta con el método **execution()** de la instancia PDOStatement aportando únicamente los valores de los atributos esperados en un array.

Ver: https://www.php.net/manual/es/pdo.prepare.php y https://www.php.net/manual/es/pdostatement.execute.php

/*Ejemplo de SENTENCIA PREPARADA*/

```
<?php
require_once __DIR__.'\..\db.php';
try {
    $db = new PDO("mysql:host=127.0.0.1:33006;dbname=EMPRESA", USERNAME, PASSWORD);
} catch (PDOException $e) {
    die("Conexión fallida: " . $e->getMessage());
}
if (isset($_POST['btnEnviar'])) {
    $SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;"); //También
podemos usar PROVEEDOR=:proveedor
    if (isset($_POST['proveedor'])){
        $SQLstring->execute(array($_POST['proveedor']));//También se puede usar BindParam
        $result=$SQLstring->fetchAll(); //Obtenemos los productos
        $SQLCount=$db->prepare("SELECT COUNT(*) FROM PRODUCTO WHERE PROVEEDOR=?;");
        $SQLCount->execute(array($_POST['proveedor'])); //También se puede usar BindParam
        $numRows=$SQLCount->fetchColumn(); //obtenemos el número de filas devueltas
        if($numRows){
            print "Se han obtenido " . $numRows . " filas." . "<br>";
            print "<h2>Productos del proveedor ".$_POST['proveedor']."</h2><br>";
            print "<table border='1'>";
            print "<tr><th>COD_PROD</th><th>NOMBRE</th><th>PROVEEDOR</th><th>PVP</th></tr>";
            foreach ($result as $row){
                print "<tr><td>$row[COD_PROD]</td><td>$row[NOMBRE]</td><td>$row[PROVEEDOR]</td><td>$row[PVP]</td></tr>";
            }
            print "</table>";
        } else {
            print "No se han obtenido resultados a mostar!\n";
        }
    } else{
        print "<p>introduce un proveedor</p>";
    }
}    ?>
```

/*Ejemplo de SENTENCIA PREPARADA*/

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Consulta de productos por proveedor</title>
</head>
<body>
    <form action="sentenciaPreparada.php" method="post">
        <h1>Consulta de productos por proveedor</h1>
        <label for="proveedor">Proveedor:</label>
        <select name="proveedor" id="proveedor">
            <?php
            /* Cargamos el select con los datos de los proveedores con productos*/
            $SQLstring = $db->prepare("SELECT DISTINCT PROVEEDOR FROM PRODUCTO;");
            $SQLstring->execute();
            $result = $SQLstring->fetchAll();
            foreach ($result as $row) {
                print "<option value='$row[PROVEEDOR]'>$row[PROVEEDOR]</option>";
            }
            ?>
            <input type="submit" name="btnEnviar" value="Enviar">
    </form>
</body>
```

/*Ejemplo de SENTENCIA PREPARADA*/

Se han obtenido 3 filas.

Productos del proveedor T98723467

COD_PROD	NOMBRE	PROVEEDOR	PVP
P0004	ALTAVOCES	T98723467	15.50
P0006	MINI PC	T98723467	200.99
P0009	PC PORTÁTIL	T98723467	750.99

Consulta de productos por proveedor

Proveedor: A12345678 Enviar

Obtención de resultados

En el ejemplo, los valores de los parámetros de la consulta SQL los hemos pasado usando un vector en el método execute (método lazy).

```
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;");
$SQLstring->execute(array($_POST['proveedor']))
```

Otro modo de pasar los valores a los parámetros es usando el método **PDOStatement::bindParam()** [o **bindValue**] y luego llamar a execute sin parámetros:

```
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;");
$SQLstring->BindParam(1,$_POST['proveedor'],PDO::PARAM_STR); //1-posición param.
$SQLstring->execute();
// O también si usamos parámetro con :nombre y no con ?
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=:proveedor;");
$SQLstring->BindParam(:proveedor,$_POST['proveedor'],PDO::PARAM_STR);
$SQLstring->execute();
```

Ver: https://www.php.net/manual/es/pdostatement.bindparam y https://www.php.net/manual/es/pdostatement.bindvalue.php

Uso de conjuntos de resultados

El conjunto de tuplas resultado se obtiene con fetch(). En el ejemplo con fetchAll():

```
array (size=3)
  0 =>
array (size=8)
      'COD_PROD' => string 'P0004' (length=5)
      0 => string 'P0004' (length=5)
      'NOMBRE' => string 'ALTAVOCES' (length=9)
      1 => string 'ALTAVOCES' (length=9)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '15.50' (length=5)
      3 => string '15.50' (length=5)
  1 =>
    array (size=8)
      'COD_PROD' => string 'P0006' (length=5)
      0 => string 'P0006' (length=5)
      'NOMBRE' => string 'MINI PC' (length=7)
      1 => string 'MINI PC' (length=7)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '200.99' (length=6)
      3 => string '200.99' (length=6)
  2 =>
    array (size=8)
      'COD_PROD' => string 'P0009' (length=5)
      0 => string 'P0009' (length=5)
      'NOMBRE' => string 'PC PORTÁTIL' (length=12)
      1 => string 'PC PORTÁTIL' (length=12)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '750.99' (length=6)
      3 => string '750.99' (length=6)
```

fetchAll() obtiene todas las filas completas del resultado de la consulta.

fetchColumn(): Devuelve sólo la columna indicada en el índice (0 por defecto, la primera) de la fila siguiente.

Ver: https://www.php.net/manual/es/pdostatement.fetch

Otros orígenes de datos - Ficheros

Las operaciones sobre ficheros suelen constar de tres fases:

1) Apertura del fichero Ver: https://www.php.net/manual/es/function.fopen.php

Se abre el fichero, indicando el modo (lectura, escritura o ambas). La función fopen() devuelve un descriptor de fichero en caso de éxito o false en otro caso.

```
<?php
    $df = fopen("c:\\folder\\resource.gif", "r"); //en Windows
    $df = fopen("/home/silvia/fichero.txt", "wb"); //sistemas Linux
    $df = fopen("http://www.example.com/", "r"); //fichero en red
?>
```

2) Procesamiento del fichero

Se realizarán las operaciones de lectura, escritura o ambas según permita la apertura del archivo. Tras finalizar, se debe liberar el descriptor para desbloquear el archivo

3) Cierre del fichero Ver: https://www.php.net/manual/es/function.fclose.php

Se cierra el fichero, devolviendo true en caso de éxito o false en caso de error

```
<?php
    fclose($df);
?>
```

Ficheros completos en una vez I

Leer y escribir un fichero completo a la vez: **file_get_contents() (lectura como string) y file_put_contents() (escritura)**

```
//Lectura del archivo plantilla.html obtenido completamente en una variable
$page = file_get_contents('plantilla.html'); //Obtiene el fichero html como string
$page = str_replace('{page_title}', 'Bienvenida', $page); // Indica título de la página
if (date('H' >= 12)) { //color de la página según si es por la mañana o la tarde
    $page = str_replace('{color}', 'blue', $page); // Azul para la mañana
} else {
    $page = str_replace('{color}', 'green', $page); //Verde para la tarde
}
// Escribimos la plantilla personalizada en un archivo llamado paginaSaludo.html
file_put_contents('paginaSaludo.html', $page);
```

Nota: en las plantillas HTML las variables entre llaves como {variable} son procesadas por php con el valor indicado en el script

fichero plantilla.html

```
<html>
    <head><title>{page_title}</title></head>
    <body bgcolor="{color}">
        <h1>Hola, eres bienvenido/a!!</h1>
    </body>
</html>
```

Ver: https://www.php.net/manual/es/function.file-get-contents.php
y https://www.php.net/manual/es/function.file-put-contents.php

Ficheros completos en una vez II

Obtener un fichero completo a la vez en un **array** donde cada elemento es una fila del fichero: **file()**

Las opciones disponibles en flags son:

FILE_USE_INCLUDE_PATH: Buscar el fichero en include_path

FILE_IGNORE_NEW_LINES: Omitir nueva línea al final de cada elemento del array

FILE_SKIP_EMPTY_LINES: Saltar las líneas vacías

```
//Obtenemos un fichero en un vector y recorremos los elementos del mismo para obtener cada una de las lineas del fichero e imprimir una lista de usuarios con su email
$vfile=file('usuarios.txt');
foreach ($vfile as $line) {
    $line = trim($line);
    $info = explode('-', $line);
    print '<li><b>' . $info[0] . '</b> tiene el email <b>' . $info[1] ."</b></li>\n";
}
```

fichero usuarios.txt

Silvia Vilar - silvia.vilar@example.com
Ana Andrés - ana.andres@example.com
Berto Barea - berto.barea@example.com
Carlos Calvo - carlos.calvo@example.com
David Díaz - david.diaz@example.com

- **Silvia Vilar** tiene el email **silvia.vilar@example.com**
- **Ana Andrés** tiene el email **ana.andres@example.com**
- **Berto Barea** tiene el email **berto.barea@example.com**
- **Carlos Calvo** tiene el email **carlos.calvo@example.com**
- **David Díaz** tiene el email **david.diaz@example.com**

Nota: resultado mostrado en el navegador

Ver: https://www.php.net/manual/es/function.file.php

Ficheros de forma parcial I

En este caso debemos llamar a fopen() ya que necesitamos el descriptor del fichero para operar con él. Veamos distintas funciones:

Función fgets(): obtiene la línea de fichero hasta alcanzar $length o eof

```
$df = fopen("usuarios.txt", "r");
while (!feof($df)){ //Mientras no alcancemos el final del archivo
    $linea = fgets($df); // $length es parámetro opcional, por defecto hasta EOF
    echo "USUARIO ",$linea, "</br>";
}
fclose($df);
```

Función fread(): obtiene conjunto de datos de ficheros binarios del tamaño indicado. Se suele indicar $length (filesize() para todo el fichero)

```
$df = fopen("usuarios.txt", "rb");
$datos = fread($df,filesize("usuarios.txt")); // leemos el fichero entero con filesize
var_dump($datos);
fclose($df);
```

Ver:
https://www.php.net/manual/es/function.fgets.php
https://www.php.net/manual/es/function.fread.php

fichero usuarios.txt

Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com

Ficheros de forma parcial II

Función fscanf(): obtiene la línea de fichero con un formato dado

```
$df = fopen("usuarios.txt", "r");
while ($usuarioinfo = fscanf($df, "%s\t%s\t%s\t%s")){
    list($usuario, $email) = $usuarioinfo;
    echo $usuarioinfo[0] . " ".$usuarioinfo[1] . " tiene el email $usuarioinfo[3] <br>";
}
fclose($df);
```

Función fseek(): lee desde la posición exacta que se le indique

```
$df = fopen("usuarios.txt", "r");
fseek($df, 40); //comenzamos a leer en el inicio de la segunda línea (segundo usuario)
while(!feof($df)){
    $linea = fgets($df);
    echo $linea, "</br>";
}
fclose($df);
```

fichero usuarios.txt

Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com

Ver:
https://www.php.net/manual/es/function.fscanf.php
https://www.php.net/manual/es/function.fseek.php

Otros orígenes de datos – Ficheros CSV

Función fgetcsv(): Busca campos csv en el fichero y devuelve un array con ellos

```
$numfila = 1; // la primera fila es el encabezado
if(($df = fopen("usuarios.csv", "r")) !== FALSE) {
    while (($fila = fgetcsv($df,0,",")) !== FALSE) { // 0 es sin límite de longitud
        if ($numfila>1){ //La primera fila son los nombres de campos del CSV
            echo "\nEl usuario $fila[0] tiene el email $fila[1] <br>";
        }
        $numfila++;
    }
fclose($df);
}
```

```
El usuario Silvia Vilar tiene el email silvia.vilar@example.com
El usuario Ana Andrés tiene el email ana.andres@example.com
El usuario Berto Barea tiene el email berto.barea@example.com
El usuario Carlos Calvo tiene el email carlos.calvo@example.com
El usuario David Díaz tiene el email david.diaz@example.com
```

fichero usuarios.csv

```
Nombre, Email
Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com
```


Ver: https://www.php.net/manual/es/function.fgetcsv.php

Otros orígenes de datos - XML

Tenemos diversas maneras de obtener documentos XML en PHP (se requiere la extensión libxml, habilitada por defecto) como por ejemplo:

- **Parser** o Analizador de **XML**: Permite analizar documentos XML (pero no validarlos, si el documento no está bien formado, da error). Es más rápido porque no carga todo el documento en una vez, analiza nodo por nodo. Ver: https://www.php.net/manual/es/book.xml.php
- Usando la API **DOM** (Document Model Object) de PHP: Se basa en análisis en árbol cargando el documento XML entero, Permite construir, modificar, consultar, validar y transformar documentos XML. Ver: https://www.php.net/manual/es/book.dom.php
- Usando **SimpleXML**: SimpleXML permite añadir, modificar, comparar elementos, usar Xpath, etc. de forma rápida y fácil. En pocas ocasiones para algo un poco más complejo puede ser necesario usar PHP DOM. Existen dos funciones para pasar de uno a otro: de un nodo DOM a un objeto SimpleXML **_simplexml_importdom()** y de un objeto SimpleXML a un nodo DOM **_dom_importsimplexml()**.Ver: https://www.php.net/manual/es/book.simplexml.php

Usuarios.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE usuarios SYSTEM "usuarios.dtd"> //Esta línea sólo es necesaria para DOM
<usuarios>
   <usuario>
      <nombre>Silvia</nombre>
      <apellido>Vilar</apellido>
      <direccion>Calle Mar 12</direccion>
      <ciudad>Valencia</ciudad>
      <pais>España</pais>
      <contacto>
         <telefono>12345678</telefono>
         <url>http://silvia.ejemplo.com</url>
         <email>silvia@ejemplo.com</email>
      </contacto>
   </usuario>
   <usuario>
      <nombre>Paco</nombre>
      <apellido>Ruiz</apellido>
      <direccion>Calle La Fuente 22</direccion>
      <ciudad>Alicante</ciudad>
      <pais>España</pais>
      <contacto>
         <telefono>23456789</telefono>
         <url>http://paco.ejemplo.com</url>
         <email>paco@ejemplo.com</email>
      </contacto>
   </usuario>
</usuarios>
```

Ejemplo de XML Parser

EjemploXMLParser.php

```
<?php
include “funcionesParser.php”;
$parser = xml_parser_create();
// Especificar el handler de elementos
xml_set_element_handler($parser,"start","stop");
// Especificar el handler de datos
xml_set_character_data_handler($parser,"char");
// Abrir un archivo xml
$df = fopen("usuarios.xml","r");
// Leer los datos
while ($data = fread($df,4096)) {
    xml_parse($parser,$data,feof($df)) or
    die (sprintf("Error XML: %s en la línea %d",
        xml_error_string(xml_get_error_code($parser)),
        xml_get_current_line_number($parser)));
}
// Liberar el analizador
xml_parser_free($parser);
?>
```

Nota: Esta forma sólo se usa en casos donde se requiera analizar el XML con rapidez y sin cargar el documento completo, aunque en este caso también se recomienda XMLReader

Ver: https://www.php.net/manual/es/book.xml.php

/*funcionesParser.php*/

```
// Función a utilizar al inicio de un elemento:
function start($parser,$elemento,$atributos) {
    switch($elemento) {
      case "USUARIO":
        echo "Datos de usuario: <br>";
        break;
      case "NOMBRE":
        echo "Nombre: ";
        break;
      case "APELLIDO":
        echo "Apellido: ";
        break;
      case "CIUDAD":
        echo "Ciudad: ";
        break;
      case "PAIS":
        echo "País: ";
            break;
      case "TELEFONO":
        echo "Teléfono: ";
        break;
      case "URL":
        echo "URL: ";
        break;
      case "EMAIL":
        echo "Email: ";
    }
}
// Función para el final de un elemento:
function stop($parser,$elemento) {
    echo "<br>";
}
// Función al encontrar un carácter
function char($parser,$data) {
    echo $data;
}
```

Ejemplo de XML DOM

```
<?php
$dom = new DOMDocument; //Instanciamos el DOM
$dom->load('usuarios.xml'); //Cargamos el fichero a analizar
$listausuarios=$dom->getElementsByTagname('usuario');
$numusuarios=$listausuarios->length;
echo "Se van a listar los datos de un total de $numusuarios usuarios: </br></br>";
foreach ($dom->getElementsByTagname('usuario') as $usuario) {
    $nombre = $usuario->getElementsByTagname('nombre')->item(0)->nodeValue;
    $apellido = $usuario->getElementsByTagname('apellido')->item(0)->nodeValue;
    $direccion = $usuario->getElementsByTagname('direccion')->item(0)->nodeValue;
    $ciudad = $usuario->getElementsByTagname('ciudad')->item(0)->nodeValue;
    $pais = $usuario->getElementsByTagname('pais')->item(0)->nodeValue;
    $telefono = $usuario->getElementsByTagname('telefono')->item(0)->nodeValue;
    $url = $usuario->getElementsByTagname('url')->item(0)->nodeValue;
    $email = $usuario->getElementsByTagname('email')->item(0)->nodeValue;
    print "Datos del usuario $nombre $apellido</br>";
    print "Dirección: $direccion</br>";
    print "Ciudad: $ciudad</br>";
    print "País: $pais</br>";
    print "Teléfono: $telefono</br>";
    print "URL: $url</br>";
    print "Email: $email</br></br>";
}
?>
```

Ver: https://www.php.net/manual/es/book.dom.php

Ejemplo de SimpleXML

```
<?php
if(!$xml = simplexml_load_file('usuarios.xml')){ //Cargamos el archivo
    echo "No se ha podido cargar el archivo";
} else {
    $numusuarios=$xml->length;
    echo "Se van a listar los datos de un total de $numusuarios usuarios:
</br></br>";
    foreach ($xml as $usuario){
        echo 'Datos del usuario '.$usuario->nombre.' '.$usuario->apellido.'<br>';
        echo 'Dirección: '.$usuario->direccion.'<br>';
        echo 'Ciudad: '.$usuario->ciudad.'<br>';
        echo 'País: '.$usuario->pais.'<br>';
        echo 'Teléfono: '.$usuario->contacto->telefono.'<br>';
        echo 'Url: '.$usuario->contacto->url.'<br>';
        echo 'Email: '.$usuario->contacto->email.'<br><br>';
    }
}
?>
```


Ver: https://www.php.net/manual/es/book.simplexml.php

Otros orígenes de datos - JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y a la vez legible para humanos (como XML, pero sin el marcado). Su sintaxis es un subconjunto del lenguaje JavaScript que fue estandarizado en 1999.
Almacena combinaciones desordenadas de **clave:valor** en **objetos { }** o utiliza **arrays [ ]** para preservar el orden de los valores, por ello es fácil de analizar y de leer. Pero también tiene limitaciones, ya que JSON sólo define una cantidad pequeña de tipos de datos, por lo que para transmitir tipos como fechas deben ser transformadas en un string o en un unix timestamp como un integer.
Los tipos de datos que soporta JSON son: strings, números, booleanos y null, además de soportar objetos y arrays como valores.

Ver: https://www.php.net/manual/en/book.json.php

Usuarios.json

```
{"usuarios":
    [
        {
            "Nombre": "Silvia",
            "Apellido": "Vilar",
            "Direccion":  "Calle Mar 12",
            "Ciudad": "Valencia",
             "Pais": "España",
             "Contacto": [
                    {
                    "Telefono": "12345678",
                    "URL": "http://silvia.ejemplo.com",
                     "Email": "silvia@ejemplo.com"
                     }
             ]
        },
        {
            "Nombre": "Paco",
            "Apellido": "Ruiz",
            "Direccion":  "Calle La Fuente 22",
            "Ciudad": "Alicante",
            "Pais": "España",
            "Contacto": [
                    {
                    "Telefono": "23456789",
                    "URL": "http://paco.ejemplo.com",
                    "Email": "paco@ejemplo.com"
                    }
            ]
        }
    ]
}
```

Otros orígenes de datos - JSON

```
<?php
$data = file_get_contents("usuarios.json");
$json = json_decode($data); // Recoge el fichero json usuarios
$usuarios=$json->usuarios; //Cargamos los usuarios (primera etiqueta json)

$numusuarios=count($usuarios);
echo "Se van a listar los datos de un total de $numusuarios usuarios: </br></br>";
foreach ($usuarios as $usuario) { //accedemos a cada usuario
    echo 'Datos del usuario '.$usuario->Nombre.' '.$usuario->Apellido.'<br>';
    echo 'Dirección: '.$usuario->Direccion.'<br>';
    echo 'Ciudad: '.$usuario->Ciudad.'<br>';
    echo 'País: '.$usuario->Pais.'<br>';
    $contacto=$usuario->Contacto[0]; //Debemos acceder al nivel de contacto
    echo 'Teléfono: '.$contacto->Telefono.'<br>';
    echo 'URL: '.$contacto->URL.'<br>';
    echo 'Email: '.$contacto->Email.'<br><br>';
}
?>
```

{objeto**}** es un objeto JSON y puedes acceder a sus propiedades directamente: $objeto->propiedad.
[objeto**]** es un array JSON y puedes acceder a sus objetos/propiedades usando un bucle o indicando el índice, por ejemplo: $objeto[0]->propiedad.

JSON (json_decode)

La función **json_decode**($json) convierte un JSON a un vector asociativo o a un objeto. Esto depende de la opción ***associative*** que cuando su valor es ***false***, convierte el JSON a un **objeto** en lugar de a un vector asociativo. Si se especifica true, se devuelve el JSON como **vector asociativo**. Si no se indica nada (associative es null) depende de si la constante JSON_OBJECT_AS_ARRAY está establecida en los flags de la función json_decode()

```
<?php
$data = file_get_contents("usuarios.json");
$vector=json_decode($data,true);
$objeto=json_decode($data,false);

echo "\n<br>JSON decodificado como vector asociativo: \n<br>";
var_dump($vector);
echo "\n<br>JSON decodificado como objeto: \n<br>";
var_dump($objeto);
}
?>
```

JSON (json_decode) - Resultado

JSON decodificado como vector asociativo:

array(1) { ["usuarios"]=> array(2) { [0]=> array(6) { ["Nombre"]=> string(6) "Silvia" ["Apellido"]=> string(5) "Vilar" ["Direccion"]=> string(12) "Calle Mar 12" ["Ciudad"]=> string(8) "Valencia" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> array(3) { ["Telefono"]=> string(8) "12345678" ["URL"]=> string(25) "http://silvia.ejemplo.com" ["Email"]=> string(18) "silvia@ejemplo.com" } } } [1]=> array(6) { ["Nombre"]=> string(4) "Paco" ["Apellido"]=> string(4) "Ruiz" ["Direccion"]=> string(18) "Calle La Fuente 22" ["Ciudad"]=> string(8) "Alicante" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> array(3) { ["Telefono"]=> string(8) "23456789" ["URL"]=> string(23) "http://paco.ejemplo.com" ["Email"]=> string(16) "paco@ejemplo.com" } } } } }

JSON decodificado como objeto:

object(stdClass)#10 (1) { ["usuarios"]=> array(2) { [0]=> object(stdClass)#6 (6) { ["Nombre"]=> string(6) "Silvia" ["Apellido"]=> string(5) "Vilar" ["Direccion"]=> string(12) "Calle Mar 12" ["Ciudad"]=> string(8) "Valencia" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> object(stdClass)#7 (3) { ["Telefono"]=> string(8) "12345678" ["URL"]=> string(25) "http://silvia.ejemplo.com" ["Email"]=> string(18) "silvia@ejemplo.com" } } } [1]=> object(stdClass)#8 (6) { ["Nombre"]=> string(4) "Paco" ["Apellido"]=> string(4) "Ruiz" ["Direccion"]=> string(18) "Calle La Fuente 22" ["Ciudad"]=> string(8) "Alicante" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> object(stdClass)#9 (3) { ["Telefono"]=> string(8) "23456789" ["URL"]=> string(23) "http://paco.ejemplo.com" ["Email"]=> string(16) "paco@ejemplo.com" } } } } }

Buenas prácticas

Es altamente recomendable que todos los datos de configuración que sean susceptibles de cambiar en los distintos entornos (desarrollo, producción, test, etc.) se separen a uno o varios ficheros de configuración

Estos ficheros pueden codificarse en PHP o utilizar otro formato que después pueda ser leído por PHP

En el ejemplo, utilizaremos un fichero config.php

Buenas prácticas - Ejemplo

Contenido de BDConfig.php

```
<?php
const HOST = 'localhost';
const DBNAME = 'EMPRESA';
const USERNAME = 'dwes';
const PASSWORD = 'dbdwespass';
$options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
?>
```

Contenido del script BDForm.php

```
<?php
require_once "BDConfig.php";
try{
$mbd = new PDO('mysql:host=HOST;dbname=DBNAME', USERNAME,
PASSWORD);
echo "Conectado con éxito en la BD ".DBNAME." alojada en ".HOST." con el
usuario ".USERNAME."\n";
} catch (PDOException $e) {
        print "Error al conectar con la BD: " .  $e->getMessage();
}
?>
```


Conectado con éxito en la BD EMPRESA alojada en localhost:33006 con el usuario dwes