UT4 -UTILIZACIÓN DE TÉCNICAS DE ACCESO A DATOS

Desarrollo Web en Entorno Servidor

2° DAW

Acceso a bases de datos desde PHP

- PHP soporta más de 15 **sistemas gestores de bases de datos:** SQLite, Oracle, SQL Server, PostgreSQL, IBM DB2, MySQL, etc
- Hasta la versión 5 de PHP, el acceso a las bases de datos se hacía principalmente utilizando extensiones específicas para cada sistema gestor de base de datos (**extensiones nativas**).
- A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: **PDO**.
- La gran ventaja de PDO es que podemos seguir utilizando una misma sintaxis aunque cambiemos el motor de nuestra base de datos.
- Mientras PDO ofrece un conjunto común de funciones, las extensiones nativas normalmente ofrecen más potencia (acceso a funciones específicas de cada gestor de base de datos) y en algunos casos también mayor velocidad.

MySQL/MariaDB

- MySQL es un gestor de bases de datos relacionales de código abierto bajo licencia GNU GPL. Es el gestor de bases de datos más empleado con el lenguaje PHP.
- Para acceder usaremos **PDO** y la extensión nativa **MySQLi**
- Incorpora múltiples motores de almacenamiento:
  - **MyISAM** (**Aria** en MariaDB): motor que se utiliza por defecto. Muy rápido pero a cambio no contempla integridad referencial ni tablas transaccionales.
  - **InnoDB** (**XtraDB** en MariaDB): es un poco más lento pero sí soporta tanto integridad referencial como tablas transaccionales.

Hoja04_BBDD_01 (Ej 1 y 2)

MySQLi

- Esta extensión se desarrolló para aprovechar las ventajas que ofrecen las versiones 4.1.3 y posteriores de MySQL, y viene incluida con PHP a partir de la versión 5.
- Ofrece un interface de programación **dual**, pudiendo accederse utilizando **objetos** o **funciones**.
  - Por ejemplo, para establecer una conexión con un servidor MySQL y consultar su versión, podemos utilizar cualquiera de las siguientes formas:

```
//utilizando constructores y métodos de la programación orientada a objetos
$conexion = new mysqli('localhost', 'usuario', 'contraseña', 'base_de_datos');
print $conexion->server_info;

// utilizando llamadas a funciones
$conexion = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

MySQLi: establecimiento de conexiones

- Si utilizas la extensión MySQLi, establecer una conexión con el servidor significa crear una instancia de la clase **mysqli**.
- El constructor de la clase puede recibir seis parámetros, todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:
  - El nombre o dirección IP del servidor MySQL al que te quieres conectar.
  - Un nombre de usuario con permisos para establecer la conexión.
  - La contraseña del usuario.
  - El nombre de la base de datos a la que conectarse.
  - El número del puerto en que se ejecuta el servidor MySQL.
  - El socket o la tubería con nombre (named pipe) a usar.
- Si se utiliza el constructor de la clase, para conectarnos a la base de datos "dwes“ se puede hacer:

```
// utilizando el constructor de la clase
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
```

- Aunque también existe la opción de primero crear la instancia, y después utilizar el método **connect** para establecer la conexión con el servidor:

```
// utilizando el método connect
$dwes = new mysqli();
$dwes->connect('localhost', 'dwes', 'abc123.', 'dwes');
```

MySQLi: establecimiento de conexiones

- Es importante verificar que la conexión se ha establecido correctamente. Para comprobar el error, en caso de que se produzca, puedes usar la propiedad de la clase mysqli:
  - **connect_errno**
- Por ejemplo, el siguiente código comprueba el establecimiento de una conexión con la base de datos "dwes" y finaliza la ejecución si se produce algún error:

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$error = $dwes->connect_errno;
if ($error != null)
{
        echo "<p>Error $error conectando a la base de datos: $dwes->connect_error</p>";
        exit();
}
```

MySQLi: establecimiento de conexiones

- Si una vez establecida la conexión, se quiere cambiar la base de datos se puede usar el método **select_db** (o la función **mysqli_select_db** de forma equivalente) para indicar el nombre de la nueva.

```
//utilizando el método connect
$dwes->select_db('otra_bd');
```

- La función mysqli_select_db, toma dos parámetros: el nombre de la base de datos, y opcionalmente, el nombre de la conexión de la base de datos. Si no se especifica este último, se emplea por defecto la última conexión que empleó mysqli_connect.

- Una vez finalizadas las tareas con la base de datos, utiliza el método **close** (o la función **mysqli_close**) para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

```
$dwes->close();
```

Hoja04_BBDD_01 (Ej 3)

MySQLi: ejecución de consultas

- La forma más inmediata de ejecutar una consulta es el método **query**, equivalente a la función **mysqli_query**.
  - Si se ejecuta una **consulta de acción** que no devuelve datos (como una sentencia SQL de tipo UPDATE, INSERT o DELETE), la llamada devuelve true si se ejecuta correctamente o false en caso contrario.
  - El número de registros afectados se puede obtener con la propiedad **affected_rows** (o con la función **mysqli_affected_rows**).

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$error = $dwes->connect_errno;
if ($error == null)
{
    $resultado = $dwes->query('DELETE FROM stock WHERE unidades=0');
    if ($resultado)
    {
            print "<p>Se han borrado $dwes->affected_rows registros.</p>";
    }
    $dwes->close();
}
```

MySQLi: transacciones

- Una transacción es una secuencia de operaciones realizadas como una sola unidad lógica de trabajo. Una unidad lógica de trabajo debe exhibir cuatro propiedades, conocidas como propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID), para ser calificada como transacción.
- Si necesitas utilizar transacciones deberás asegurarte de que estén soportadas por el motor de almacenamiento que gestiona tus tablas en MySQL. Si utilizas InnoDB, por defecto cada consulta individual se incluye dentro de su propia transacción. Puedes gestionar este comportamiento con el método **autocommit** (función **mysqli_autocommit**).

```
$dwes->autocommit(false);
```

- Al deshabilitar las transacciones automáticas, las siguientes operaciones sobre la base de datos iniciarán una transacción que deberás finalizar utilizando:
  - **commit** (o la función mysqli_commit). Realizar una operación "commit" de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.
  - **rollback** (o la función mysqli_rollback). Realizar una operación "rollback" de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.

```
...
// Inicia una transacción
$dwes->query('DELETE FROM stock WHERE unidades=0');
$dwes->query('UPDATE stock SET unidades=3 WHERE producto="STYLUSSX515W"');
...
$dwes->commit();
// Confirma los cambios
```

- Una vez finalizada esa transacción, comenzará otra de forma automática

MySQLi: obtención y utilización de conjuntos de resultados

- En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto resultado (de la clase mysqli_result).
- El método **query** tiene un parámetro opcional que afecta a cómo se obtienen internamente los resultados, pero no a la forma de utilizarlos posteriormente.
  - En la opción por defecto, MYSQLI_STORE_RESULT, los resultados se recuperan todos juntos de la base de datos y se almacenan de forma local. Si se utiliza el valor MYSQLI_USE_RESULT, los datos se van recuperando del servidor según se vayan necesitando.
  - `$resultado = $dwes->query('SELECT producto, unidades FROM stock', MYSQLI_USE_RESULT);`
- Es importante tener en cuenta que los resultados obtenidos se almacenarán en memoria mientras se estén usando. Cuando no se necesiten, se pueden liberar con el método **free** de la clase mysqli_result (o con la función mysqli_free_result):
  - `$resultado->free();`
- Para poder obtener las filas a partir del conjunto de resultados, se utiliza **mysqli_fetch_row**, que toma como parámetro el resultado de **mysqli_result**.
  - Devuelve una fila a la vez desde la consulta, hasta que no hay más filas, momento en el cual muestra FALSE. Lo que se debe hacer es ejecutar un bucle sobre el resultado de mysqli_fech_row

MySQLi: obtención y utilización de conjuntos de resultados

- Para trabajar con los datos obtenidos del servidor, hay varias posibilidades:
  - **fetch_array** (función **mysqli_fetch_array**): obtiene un registro completo del conjunto de resultados y lo almacena en un array. Por defecto el array contiene tanto claves numéricas como asociativas. Por ejemplo, para acceder al primer campo devuelto, podemos utilizar como clave el número 0 o por su nombre.

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock
  WHERE unidades<2');
$stock = $resultado->fetch_array();
// Obtenemos el primer registro
$producto = $stock['producto']; // O también $stock[0];
$unidades = $stock['unidades']; // O también $stock[1];
print "<p>Producto $producto: $unidades unidades.</p>";
```

    - Este comportamiento se puede modificar utilizando un parámetro opcional a la función fetch_array , que puede tomar los siguientes valores:
      - MYSQLI_NUM: devuelve un array con claves numéricas.
      - MYSQLI_ASSOC: devuelve un array asociativo.
      - MYSQLI_BOTH: es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.

MySQLi: obtención y utilización de conjuntos de resultados

- **fetch_assoc** (función mysqli_fetch_assoc): idéntico a fetch_array pasando como parámetro MYSQLI_ASSOC.
- **fetch_row** (función mysqli_fetch_row): idéntico a fetch_array pasando como parámetro MYSQLI_NUM.
- **fetch_object** (función mysqli_fetch_object): similar a los métodos anteriores, pero devuelve un objeto en lugar de un array. Las propiedades del objeto devuelto se corresponden con cada uno de los campos del registro.

- Parar recorrer todos los registros de un array, se puede hacer un bucle teniendo en cuenta que cualquiera de los métodos o funciones anteriores devolverá null cuando no haya más registros en el conjunto de resultados.

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock WHERE
   unidades<2');
$stock = $resultado->fetch_object();
while ($stock != null) {
   print "<p>Producto $stock->producto: $stock->unidades unidades.</p>";
   $stock = $resultado->fetch_object();
}
```

Hoja04_BBDD_01 (Ej 4, 5, 6 y 7)

MySQLi: consultas preparadas

- Cada vez que se envía una consulta al servido se analiza antes de ejecutarla. Algunas sentencias deben repetirse de forma habitual en un programa. Para acelerar este proceso, MySQL admite consultas preparadas. Estas consultas se almacenan en el servidor listas para ser ejecutadas cuando sea necesario.
- Para trabajar con consultas preparadas con la extensión MySQLi de PHP, debes utilizar la clase **mysqli_stmt**. Utilizando el método **stmt_init** de la clase mysqli (o la función mysqli_stmt_init) obtienes un objeto de dicha clase.
- Los pasos para ejecutar una consulta preparada son:
  - Preparar la consulta en el servidor MySQL utilizando el método **prepare** (función mysqli_stmt_prepare).
  - Ejecutar la consulta, tantas veces como sea necesario, con el método **execute** (función mysqli_stmt_execute).
  - Una vez que ya no se necesita más, se debe ejecutar el método **close** (función mysqli_stmt_close).
- Por ejemplo, para preparar y ejecutar una consulta que inserta un nuevo registro:

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$consulta = $dwes->stmt_init();
$consulta->prepare('INSERT INTO alumno (cod, nombre) VALUES ("daw200", "Iván")');
$consulta->execute();
$consulta->close();
$dwes->close();
```

MySQLi: consultas preparadas

- ¿Alguien ve algún **problema**?
  - ¿Tiene algún sentido preparar una consulta si los datos son siempre los mismos…?
- Por este motivo las consultas preparadas admiten **parámetros.**
- Para preparar una consulta con parámetros, en lugar de poner los valores hay que indicar con un signo de interrogación su posición dentro de la sentencia SQL.

```
$consulta->prepare('INSERT INTO alumno (cod, nombre) VALUES (?,?)');
```

- Y antes de ejecutar la consulta hay que utilizar el método **bind_param** (o la función mysqli_stmt_bind_param) para sustituir cada parámetro por su valor.

```
$consulta->prepare('INSERT INTO alumno (cod, nombre) VALUES (?,?)');
$cod_alumno = "daw201";
$nombre_producto = "Iván";
$consulta->bind_param('ss', $cod_producto, $nombre_producto);
```

MySQLi: consultas preparadas

- El primer parámetro del método bind_param es una cadena de texto en la que cada carácter indica el tipo de un parámetro, según la siguiente tabla.

Carácter	Tipo de parámetro
i	Número entero
d	Número real (doble precisión)
s	Cadena de texto
b	Contenido en formato binario

- En el caso de las consultas que devuelven valores, se puede utilizar el método **bind_result** (función mysqli_stmt_bind_result) para asignar a variables los campos que se obtienen tras la ejecución. Utilizando el método **fetch** (mysqli_stmt_fetch) se recorren los registros devueltos:

```
$consulta = $dwes->stmt_init();
$consulta->prepare('SELECT campo1, campo2 FROM tabla WHERE condicion');
$consulta->execute();
$consulta->bind_result($campo1, $campo2);
while($consulta->fetch())
{
      echo $campo1.": ".$campo2."<br />";
}
$consulta->close();
```

Hoja04_BBDD_01 (Ej 8)

PHP Data Objects (PDO)

- ¿Y si queremos cambiar en el futuro el SGBD por otro distinto? ¿Volvemos a programar gran parte del código?
  - Antes de comenzar el desarrollo hay que revisar las características específicas del proyecto. En el caso de que exista la posibilidad de utilizar otro servidor como almacenamiento, hay que adoptar una capa de abstracción para el acceso a los datos.
  - Existen varias alternativas como ODBC, pero sin duda la opción más recomendable en la actualidad es **PDO**.
- PDO se basa en las características de orientación a objetos de PHP pero, al contrario que la extensión MySQLi, no ofrece un interface de programación dual.
- Para acceder a las funcionalidades de la extensión hay que emplear los objetos que ofrece, con sus métodos y propiedades. No existen funciones alternativas.

PDO: establecimiento de conexiones

- Para establecer una conexión con una base de datos utilizando PDO hay que instanciar un objeto de la clase PDO pasándole los siguientes parámetros (solo el primero es obligatorio):
  - Origen de datos (DSN). Es una cadena de texto que indica qué controlador se va a utilizar y a continuación, separadas por el carácter dos puntos, los parámetros específicos necesarios por el controlador, como por ejemplo el nombre o dirección IP del servidor y el nombre de la base de datos.
  - Nombre de usuario con permisos para establecer la conexión.
  - Contraseña del usuario.
  - Opciones de conexión, almacenadas en forma de array.

```
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'abc123.');
```

- Si se utiliza el controlador para MySQL, los parámetros específicos para utilizar en la cadena DSN (separadas unas de otras por el carácter punto y coma) a continuación del prefijo mysql: son los siguientes:
  - **host**: nombre o dirección IP del servidor.
  - **port**: número de puerto TCP en el que escucha el servidor.
  - **dbname**: nombre de la base de datos.
  - **unix_socket**: socket de MySQL en sistemas Unix.

- Para indicar que utilice codificación UTF-8 para los datos que se transmitan:
  - `$opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");`
  - `$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'abc123.', $opciones);`

Hoja04_BBDD_01 (Ej 3 con PDO)

PDO: ejecución de consultas

- Hay que diferenciar entre las sentencias SQL que no devuelven datos, de aquellas que sí lo devuelven.
- En el caso de las consultas de acción, como INSERT, DELETE o UPDATE, el método **exec** devuelve el número de registros afectados.

```
$registros=$dwes->exec('DELETE FROM stock WHERE unidades=0');
echo "<p>Se han borrado $registros registros.</p>";
```

- Si la consulta genera un conjunto de datos (SELECT) hay que utilizar el método **query**, que devuelve un objeto de la clase **PDOStatement.**

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");
$resultado = $dwes->query("SELECT producto, unidades FROM stock");
```

PDO: transacciones

- Por defecto PDO trabaja en modo "autocommit", es decir, que confirma de forma automática cada sentencia que ejecuta el servidor. Para trabajar con transacciones, PDO incorpora tres métodos:
  - **beginTransaction**: deshabilita el modo "autocommit" y comienza una nueva transacción, que finalizará cuando se ejecute uno de los dos métodos siguientes.
  - **commit**: confirma la transacción actual.
  - **rollback**: revierte los cambios llevados a cabo en la transacción actual.

- Una vez ejecutado un commit o un rollback, se volverá al modo de confirmación automática.

```
$ok = true;
$dwes->beginTransaction();
if($dwes->exec('DELETE …') == 0)  $ok = false;
if($dwes->exec('UPDATE …') == 0)  $ok = false;
…
if ($ok) $dwes->commit();  // Si todo fue bien confirma los cambios
else $dwes->rollback(); // y si no, los revierte
```

PDO: obtención y utilización de conjuntos de resultados

- Al igual que en MySQLi, en PDO hay varias posibilidades para tratar con el conjunto de resultados devueltos por el método query. La más utilizada es el método fetch:

```
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE condicion');
while ($registro = $resultado->fetch())
{
echo $registro['campo1'].": ".$registro['campo2']."<br />";
}
```

- Por defecto, el método **fetch** genera y devuelve, a partir de cada registro, un array con claves numéricas y asociativas. Para cambiar su comportamiento, admite un parámetro opcional que puede tomar uno de los siguientes valores:
  - PDO::FETCH_ASSOC: devuelve solo un array asociativo.
  - PDO::FETCH_NUM: devuelve solo un array con claves numéricas.
  - PDO::FETCH_BOTH: devuelve un array con claves numéricas y asociativas. Es el comportamiento por defecto.

PDO: obtención y utilización de conjuntos de resultados

- PDO::FETCH_OBJ: devuelve un objeto cuyas propiedades se corresponden con los campos del registro.

```
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE condicion');
while ($registro = $resultado->fetch(PDO::FETCH_OBJ))
{
        echo $registro->campo1.": ".$registro->campo2."<br />";
}
```

- PDO::FETCH_LAZY: devuelve tanto el objeto como el array con clave dual anterior.
- PDO::FETCH_BOUND: devuelve true y asigna los valores del registro a variables, según se indique con el método bindColumn. Este método debe ser llamado una vez por cada columna, indicando en cada llamada el número de columna (empezando en 1) y la variable a asignar.

```
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE condicion');
$resultado->bindColumn(1, $campo1);
$resultado->bindColumn(2, $campo2);
while ($registro = $resultado->fetch(PDO::FETCH_BOUND))
{
        echo $campo1.": ".$campo2."<br />";
}
```

Hoja04_BBDD_01 (Ej 4, 5, 6 y 7 con PDO)

PDO: consultas preparadas

- Para preparar la consulta se utiliza el método **prepare** de la clase PDO. Este método devuelve un objeto de la clase **PDOStatement**. Los parámetros se pueden marcar utilizando signos de interrogación como en MySQLi.

```
$consulta=$dwes->prepare('INSERT INTO tabla (campo1, campo2) VALUES (?, ?)');
```

- O también se pueden utilizar parámetros con nombre, precediéndolos por el símbolo de dos puntos:

```
$consulta=$dwes->prepare('INSERT INTO tabla (campo1, campo2) VALUES (:cm1, :cm2)');
```

- Si se utiliza parámetros con nombre hay que indicar ese nombre en la llamada a **bindParam**

```
$campo1 = "valor_campo1";
$campo2 = "valor_campo2";
$consulta->bindParam(1, $campo1); //o $consulta->bindParam(:cm1, $campo1);
$consulta->bindParam(2, $campo2); //o $consulta->bindParam(:cm2, $campo2);
```

- Una vez preparada la consulta y enlazados los parámetros con sus valores, se ejecuta la consulta utilizando el método **execute**.

```
$consulta->execute();
```

Hoja04_BBDD_01 (Ej 8 con PDO)

Errores y manejo de excepciones

- PHP define una **clasificación de los errores** que se pueden producir en la ejecución de un programa y ofrece métodos para ajustar el tratamiento de los mismos. Para hacer referencia a cada uno de los niveles de error, PHP define una serie de **constantes**.
- Cada nivel se identifica por una constante. Por ejemplo, la constante E_NOTICE hace referencia a avisos que pueden indicar un error al ejecutar el script, y la constante E_ERROR engloba errores fatales que provocan que se interrumpa forzosamente la ejecución.
- La configuración inicial de cómo se va a tratar cada error según su nivel se realiza en php.ini el fichero de configuración de PHP.
- Entre los principales parámetros que puedes ajustar están:
  - **error_reporting**: indica qué tipos de errores se notificarán. Su valor se forma utilizando los operadores a nivel de bit para combinar las constantes anteriores. Su valor predeterminado es E_ALL & ~E_NOTICE que indica que se notifiquen todos los errores (E_ALL) salvo los avisos en tiempo de ejecución (E_NOTICE).
  - **display_errors**: en su valor por defecto (On), hace que los mensajes se envíen a la salida estándar (y por lo tanto se muestren en el navegador). Se debe desactivar (Off) en los servidores que no se usan para desarrollo sino para producción.
- Desde código se puede usar la función **error_reporting** con las constantes anteriores para establecer el nivel de notificación en un momento determinado.

Errores y manejo de excepciones

- A partir de la versión 5 se introdujo en PHP un modelo de excepciones similar al existente en otros lenguajes de programación:
  - El código susceptible de producir algún error se introduce en un bloque **try**.
  - Cuando se produce algún error, se lanza una excepción utilizando la instrucción **throw**.
  - Después del bloque try debe haber como mínimo un bloque **catch** encargado de procesar el error.
  - Si una vez acabado el bloque try no se ha lanzado ninguna excepción, se continúa con la ejecución en la línea siguiente al bloque o bloques catch.
- Por ejemplo, para lanzar una excepción cuando se produce una división por cero podrías hacer:

```
try {
        if ($divisor == 0)
                throw new Exception("División por cero.");
        $resultado = $dividendo / $divisor;
}
catch (Exception $e) {
        echo "Se ha producido el siguiente error: ".$e->getMessage();
}
```

Errores y manejo de excepciones

- La clase PDO permite definir la fórmula que usará cuando se produzca un error, utilizando el atributo PDO::ATTR_ERRMODE. Las posibilidades son:
  - **PDO::ERRMODE_SILENT**: no se hace nada cuando ocurre un error. Es el comportamiento por **defecto**.
  - **PDO::ERRMODE_WARNING**: genera un error de tipo E_WARNING cuando se produce un error.
  - **PDO::ERRMODE_EXCEPTION**: cuando se produce un error lanza una excepción utilizando el manejador propio PDOException.
- Es decir, que si quieres utilizar excepciones con la extensión PDO, debes configurar la conexión haciendo:

```
$dwes->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

- Por ejemplo, el siguiente código captura la excepción que lanza PDO debido a que la tabla no existe:

```
$dwes = new PDO("mysql:host=localhost; dbname=dwes", "dwes", "abc123.");
$dwes->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
        $sql = "SELECT * FROM tablaincorrecta";
        $result = $dwes->query($sql);
        …
}
catch (PDOException $p) {
        echo "Error ".$p->getMessage()."<br />";
}
```