

Unidad 5

Acceso a BD

Desde una página con código escrito en PHP nos podemos conectar a una base de datos y ejecutar comandos

en lenguaje SQL - para hacer consultas, insertar o modificar registros, crear tablas, dar permisos, etc.

PHP puede trabajar con la práctica totalidad de gestores de bases de datos que hay disponibles, tanto

comerciales como de código abierto.

Mysqli. Conexión a BD.

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 a las funcionalidades de la extensión utilizando objetos o funciones de forma indiferente. 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. Ejecutar SELECT a BD

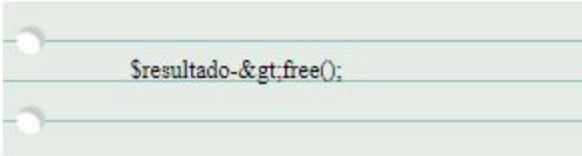
En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto resultado (de la clase `mysqli_result`).

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 cambiamos esta opción por 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);
```

Liberar memoria.

Es importante tener en cuenta que los resultados obtenidos se almacenarán en memoria mientras los estés usando. Cuando ya no los necesites, los puedes liberar con el método `free` de la clase `mysqli_result` (o con la función `mysqli_free_result`):



```
$resultado->free();
```

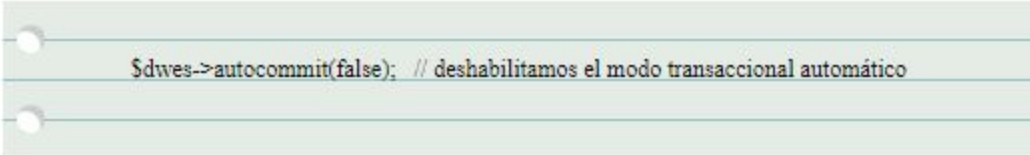
Mysqli. Ejecución de consultas. Insert, Delete y Update.

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.

Si necesitas utilizar transacciones deberás asegurarte de que estén soportadas por el motor de almacenamiento que gestiona tus tablas en MySQL. Puedes gestionar este comportamiento con el método `autocommit` (función `mysqli_autocommit`).

A screenshot of a terminal window with a light green background. It shows a single line of text: `$dwes->autocommit(false); // deshabilitamos el modo transaccional automático`. The text is in a monospaced font, and the comment part is in a lighter blue color.

```
$dwes->autocommit(false); // deshabilitamos el modo transaccional automático
```

Hay que tener en cuenta que no todas los motores de BD soportan transacciones.

Mysqli. Transacciones

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.

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

```
...  
$dwes->query("DELETE FROM stock WHERE unidades=0"); // Inicia una transacción  
$dwes->query("UPDATE stock SET unidades=3 WHERE producto='STYLUS$X515W'");  
...  
$dwes->commit(); // Confirma los cambios
```


Mysqli. Consultas Preparadas

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.

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');  
$consulta = $dwes->stmt_init();
```

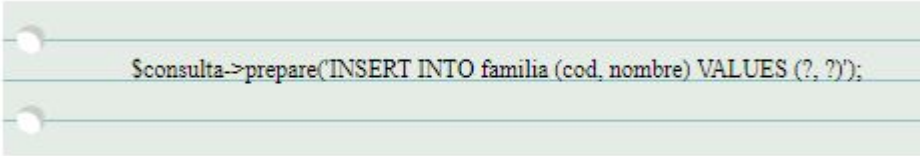
Los pasos que debes seguir para ejecutar una consulta preparada son:

- 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`).

```
$consulta = $dwes->stmt_init();  
$consulta->prepare("INSERT INTO familia (cod, nombre) VALUES ('TABLET', 'Tablet PC')");  
$consulta->execute();  
$consulta->close();  
$dwes->close();
```

Mysqli. Consultas Preparadas

El problema que ya habrás observado, es que de poco sirve preparar una consulta de inserción de datos como la anterior, si los valores que inserta son siempre los mismos. Por este motivo las consultas preparadas admiten parámetros. Para preparar una consulta con parámetros, en lugar de poner los valores debes indicar con un signo de interrogación su posición dentro de la sentencia SQL.

A screenshot of a code editor with a light green background and horizontal lines. The text `$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?);` is written in a monospaced font. The opening quote is a single quote, and the closing quote is a double quote.

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

Y antes de ejecutar la consulta tienes que utilizar el método `bind_param` (o la función `mysqli_stmt_bind_param`) para sustituir cada parámetro por su valor. 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 del parámetro.
I.	Número entero.
D.	Número real (doble precisión).
S.	Cadena de texto.
B.	Contenido en formato binario (BLOB).

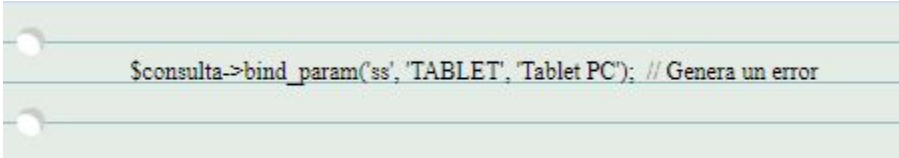
Mysqli. Consultas Preparadas

En el caso anterior, si almacenamos los valores a insertar en sendas variables, puedes hacer:

```
$consulta = $dwes->stmt_init();  
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');  
$cod_producto = "TABLET";  
$nombre_producto = "Tablet PC";  
$consulta->bind_param('ss', $cod_producto, $nombre_producto);  
$consulta->execute();  
$consulta->close();  
$dwes->close();
```

Mysqli. Consultas Preparadas

Cuando uses `bind_param` para enlazar los parámetros de una consulta preparada con sus respectivos valores, deberás usar siempre variables como en el ejemplo anterior. Si intentas utilizar literales, por ejemplo:

A screenshot of a code editor with a light green background and horizontal lines. The code is written in a monospaced font. The statement is: `$consulta->bind_param('ss', 'TABLET', 'Tablet PC'); // Genera un error`. The text is in black, with some parts in green and red to indicate syntax highlighting.

```
$consulta->bind_param('ss', 'TABLET', 'Tablet PC'); // Genera un error
```

Obtendrás un error. El motivo es que los parámetros del método `bind_param` se pasan por **referencia**. Aprenderás a usar paso de parámetros por referencia en una unidad posterior.

Mysqli. Consultas Preparadas

El método `bind_param` permite tener una consulta preparada en el servidor MySQL y ejecutarla tantas veces como quieras cambiando ciertos valores cada vez. Además, 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. Observa el siguiente código:

```
$consulta = $dwes->stmt_init();  
$consulta->prepare("SELECT producto, unidades FROM stock WHERE unidades<2");  
$consulta->execute();  
$consulta->bind_result($producto, $unidades);  
while($consulta->fetch()) {  
    print "<p>Producto $producto: $unidades unidades.</p>";  
}  
$consulta->close();  
$dwes->close();
```

Tarea

Crear una BD de usuarios para continuar con el ejercicio de login pero esta vez accediendo a la BD. Para ello accede a phpMyAdmin y crea una nueva BD llamada usuarios. En esta nueva BD crea una tabla que se llame también usuarios con la siguiente estructura:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
<input type="checkbox"/>	1	usuario	varchar(50)	utf8mb4_0900_ai_ci		No	Ninguna		
<input type="checkbox"/>	2	password	varchar(200)	utf8mb4_0900_ai_ci		No	Ninguna		
<input type="checkbox"/>	3	id 	int			No	Ninguna		AUTO_INCREMENT

Modifica el código de la práctica de login y accede a la BD para consultar usuarios y crear usuarios.

PHP Data Objects (PDO)

La extensión mysqli se puede utilizar con el formato procedimental o bien con el formato de Programación Orientada a Objetos.

PDO. Conexión a BD.

Podemos establecer una conexión con la base de datos 'dwes' creada anteriormente de la siguiente forma:

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


PDO. Ejecutar SELECT a BD

Al igual que con la extensión MySQLi, en PDO tienes varias posibilidades para tratar con el conjunto de resultados devuelto por el método `query`. La más utilizada es el método `fetch` de la clase PDOStatement. Este método devuelve un registro del conjunto de resultados, o false si ya no quedan registros por recorrer.

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$resultado = $dwes->query("SELECT producto, unidades FROM stock");  
while ($registro = $resultado->fetch()) {  
    echo "Producto ". $registro['producto']. ": ". $registro['unidades']. "<br />";  
}
```

PDO. Fetch

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::FETCH_OBJ**. Devuelve un objeto cuyas propiedades se corresponden con los campos del registro.

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$resultado = $dwes->query("SELECT producto, unidades FROM stock");  
while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {  
    echo "Producto ".$registro->producto.": ".$registro->unidades."<br />";  
}
```

PDO. Ejecución de consultas. Insert, Delete y Update.

Para ejecutar una consulta SQL utilizando PDO, debes diferenciar aquellas sentencias SQL que no devuelven como resultado un conjunto de datos, de aquellas otras 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');  
print "<p>Se han borrado $registros registros.</p>";
```

PDO. Transacciones.

Por defecto PDO trabaja en modo "autocommit", esto es, 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 ejecutes 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. Consultas Preparadas

Al igual que con MySQLi, también utilizando PDO podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida. El procedimiento es similar e incluso los métodos a ejecutar tienen prácticamente los mismos nombres.

Para preparar la consulta en el servidor MySQL, deberás utilizar 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 el caso anterior.

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$consulta = $dwes->prepare("INSERT INTO familia (cod, nombre) VALUES (?, ?);");
```

O también utilizando parámetros con nombre, precedidos por el símbolo de dos puntos.

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$consulta = $dwes->prepare("INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre);");
```

PDO. Consultas Preparadas

Antes de ejecutar la consulta hay que asignar un valor a los parámetros utilizando el método `bindParam` de la clase `PDOStatement`.. Si utilizas signos de interrogación para marcar los parámetros, el procedimiento es equivalente al método `bindColumn` que acabamos de ver.

```
$cod_producto = "TABLET";  
$nombre_producto = "Tablet PC";  
$consulta->bindParam(1, $cod_producto);  
$consulta->bindParam(2, $nombre_producto);
```

Si utilizas parámetros con nombre, debes indicar ese nombre en la llamada a `bindParam`.

```
$consulta->bindParam(":cod", $cod_producto);  
$consulta->bindParam(":nombre", $nombre_producto);
```

PDO. Consultas Preparadas

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

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

Alternativamente, es posible asignar los valores de los parámetros en el momento de ejecutar la consulta, utilizando un array (asociativo o con claves numéricas dependiendo de la forma en que hayas indicado los parámetros) en la llamada a **execute**.

```
$parametros = array(":cod" => "TABLET", ":nombre" => "Tablet PC");  
$consulta->execute($parametros);
```

Tarea

Modifica el código de la práctica de login y accede a la BD usando PDO.

Crear conexión con mysqli vs PDO

mysqli

```
// Crear una conexión a la base de datos
$conn = new mysqli($host, $dbUsername, $dbPassword,
$dbName);

// Verificar la conexión
if ($conn->connect_error) {
    die("Error de conexión: ".$conn->connect_error);
}
```

PDO

```
// Crear una conexión a la base de datos
$conn = new
PDO("mysql:host=$host;dbname=$dbName",
$dbUsername, $dbPassword);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
```

Statement con mysqli

```
//Iniciar el statement
$stmt = $conn->stmt_init();
//Preparar el statement
$stmt->prepare('SELECT * FROM tabla');
//Añadir los parámetros
$stmt->bind_param('s', $condition);
//Ejecutar el statement
$stmt->execute();
//Obtener los resultados
$resultado = $stmt->get_result();
//Comprobar si la consulta devuelve filas
if ($resultado->num_rows > 0) {
    // Mostrar los datos de cada fila
    while ($fila = $resultado->fetch_assoc()) {
        echo "ID: " . $fila["id"]. " - Nombre: " . $fila["nombre"]. " - Email: " . $fila["email"]. "<br>";
    }
} else {
    echo "La consulta no devuelve filas.";
}
```

Statement con PDO

```
try {  
    //Crear conexión  
    ...  
    //Preparar el statement  
    $statement = $conn->prepare("SELECT * FROM tabla");  
    //Ejecutar el statement  
    $statement->execute();  
    //Obtener resultados como un array asociativo  
    $resultados = $statement->fetchAll(PDO::FETCH_ASSOC);  
    //Verificar si la consulta devuelve filas  
    if (count($resultados) > 0) {  
        //Mostrar los datos de cada fila  
        foreach ($resultados as $fila) {  
            echo "ID: " . $fila["id"]. " - Nombre: " . $fila["nombre"]. " - Email: " . $fila["email"]. "<br>";  
        }  
    } else {  
        echo "La consulta no devuelve filas.";  
    }  
} catch(PDOException $e) {  
    echo "Error: " . $e->getMessage();  
}
```

Cerrar statements y conexiones

Mysqli

```
// Cerrar el statement y la conexión a la  
base de datos  
$stmt->close();  
$conn->close();
```

PDO

```
//No hace falta cerrar los statements, se hace  
automáticamente  
//Cerrar la conexión a la base de datos  
(automáticamente se cierra al salir del script)  
$conn = null;
```