Tema 4. Acceso a datos

DESARROLLO EN ENTORNO SERVIDOR 2º DAW

MARÍA CRIADO DOMÍNGUEZ

Índice

Instalar y configurar mysql

MySQL es un sistema gestor de bases de datos (SGBD) relacionales. Es un programa de código abierto que se ofrece bajo licencia GNU GPL

MySQL se emplea en múltiples aplicaciones web, ligado en la mayor parte de los casos al lenguaje PHP y al servidor web Apache. Utiliza SQL para la gestión, consulta y modificación de la información almacenada. Soporta la mayor parte de las características de ANSI SQL

En Linux, la instalación de MySQL se divide básicamente en dos paquetes que puedes instalar:

- mysql-server. Es el servidor en sí. Necesitas instalar este paquete para gestionar las bases de datos y permitir conexiones desde el equipo local o a través de la red.
- mysql-client. Es el cliente desde donde se hace la conexión con el servidor

Instalar y configurar mysql

Una vez instalado, puedes gestionar la ejecución del servicio de la misma forma que cualquier otro servicio del sistema:

```
systemctl status mysql
```

En una instalación típica, el usuario root no tiene por defecto contraseña de acceso al servidor. Es importante asignarle una por razones de seguridad:

```
mysqladmin -u root password nueva-contraseña
```

El servidor se ejecuta por defecto en el Puerto TCP 3306.

Utilización de bases de datos MySQL en PHP

Las acciones sobre las bases de datos como:

- Establecer conexiones.
- Ejecutar sentencias SQL.
- Obtener los registros afectados o devueltos por una sentencia SQL.
- Emplear transacciones.
- Ejecutar procedimientos almacenados.
- Gestionar los errores que se produzcan durante la conexión o en el establecimiento de la misma.

MySQLi

Esta extensión que 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.

```
// 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

Las opciones de configuración de PHP se almacenan en el fichero php.ini. En este fichero hay una sección específica para las opciones de configuración propias de cada extensión. Entre las opciones que puedes configurar para la extensión MySQLi están:

- mysqli.allow_persistent. Permite crear conexiones persistentes.
- mysqli.default_port. Número de puerto TCP predeterminado a utilizar cuando se conecta al servidor de base de datos.
- mysqli.reconnect. Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.
- mysqli.default_host. Host predeterminado a usar cuando se conecta al servidor de base de datos.
- mysqli.default_user. Nombre de usuario predeterminado a usar cuando se conecta al servidor de base de datos.
- mysqli.default_pw. Contraseña predeterminada a usar cuando se conecta al servidor de base de datos.

MySQLi

Para poder comunicarte desde un programa PHP con un servidor MySQL, el primer paso es establecer una conexión. Toda comunicación posterior que tenga lugar, se hará utilizando esa conexión.

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.

MySQLi

Conexión

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes’);
```

```
$dwes = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
```

Es importante verificar que la conexión se ha establecido correctamente. Para comprobar el error, en caso de que se produzca, puedes usar las siguientes propiedades (o funciones equivalentes) de la clase mysqli:

- connect_errno (o la función mysqli_connect_errno) devuelve el número de error o null si no se produce ningún error.
- connect_error (o la función mysqli_connect_error) devuelve el mensaje de error o null si no se produce ningún error.

MySQLi

Conexión

Si una vez establecida la conexión, quieres cambiar la base de datos puedes usar el método select_db (o la función mysqli_select_db de forma equivalente) para indicar el nombre de la nueva.

```
$dwes->select_db('otra_bd’);
```

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();
```

MySQLi

Ejecutar consultas

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).

```
$resultadoConsulta = $miDB->query('consultaSQL');
```

```
$resultadoConsulta = $miDB->query('consultaSQL', MYSQLI_USE_RESULT);
```

- MYSQLI_USE_RESULT //Recupera la información a medida que la usamos
- MYSQLI_STORE_RESULT //Recupera toda la información al ejecutar la consulta (defecto)

```
$miDB->affected_rows;
```

```
$resultadoConsulta->free();  //Libera el espacio ocupado tras la consulta
```

MySQLi

Obtección de resultados

```
$registroArray = $resultadoConsulta->fetch_array();
```

Obtiene un registro en el array (asociativo y numérico por defecto) y avanza el puntero por el conjunto de registros

```
$registroObjeto = $resultadoConsulta->fetch_object();
```

Obtiene un registro en el objeto (asociativo) y avanza el puntero por el conjunto de registros

Ambos métodos devuelven null cuando terminan de recorrer el conjunto de resultados.

MySQLi

Consltas preparadas (no devuelve resultados)

```
$consulta = $miDB->stmt_init(); //
```
Devuelve un objeto de la clase mysqli_stmt

```
$consulta->prepare('ConsultaPreparada VALUES (?,?,…));
```

Preparamos la consulta pendiente de dar valor a las ?

```
$consulta->bind_param('cadenaformato[i,d,s,b], $var1, $var2,…);
```

Sustituye las interrogaciones por el valor de las variables que pasamos en bind_param i – entero d – real doble precisión s – cadena de texto b – BLOB – binario

```
$consulta->execute(); //Ejecuta la consulta
$consulta->close();
$miDB->close();
```

MySQLi

Consltas preparadas (con resultados)

```
$consulta = $miDB->stmt_init(); //Devuelve un objeto de la clase mysqli_stmt
$consulta->prepare('SELECT campo1, campo2,... FROM…');
$consulta->execute(); //Ejecuta la consulta
$consulta->bind_result($var1, $var2,…);
while ($consulta->fetch())
      {Tratamiento de $var1, $var2,...}
$consulta->close();
$miDB->close();
```

MySQLi

Transacciones

```
   $miDB->autocommit(false);
 //Deshabilita el modo transaccional automático
     $resultadoConsulta1 = $miDB->query('delete from usuarios where 1’);
//si todo ha ido bien
     $miDB->commit(); //Confirma los cambios y los consolida
// sino
     $miDB->rollback(); //Revierte o deshace los cambios
```

PDO

Utilizar otro servidor como almacenamiento que no sea MySql, deberás 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.

El objetivo es que si llegado el momento necesitas cambiar el servidor de base de datos, las modificaciones que debas realizar en tu código sean mínimas. Incluso es posible desarrollar aplicaciones preparadas para utilizar un almacenamiento u otro según se indique en el momento de la ejecución.

PDO

La extensión PDO debe utilizar un driver o controlador específico para el tipo de base de datos que se utilice. Para consultar los controladores disponibles en tu instalación de PHP, puedes utilizar la información que proporciona la función phpinfo.

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 tienes que emplear los objetos que ofrece, con sus métodos y propiedades. No existen funciones alternativas.

PHP: Controladores de PDO – Manual

PDO

Establecimiento de conexión

```
$miDB = new PDO('DSN','nobreusuario','password');
```

Instanciamos un objeto PDO y establecemos la conexión

Construcción de la cadena PDO: (ej. 'mysql:host=localhost; dbname=midb')

host – nombre o dirección IP del servidor

port – número de puerto en el que escucha el servidor

dbname – nombre de la base de datos

```
$miDB->getAtribute(…);  //Obtiene el valor de un atributo
$miDB->setAtribute(…);  //Establece el valor de un atributo
unset($miDB);
```

PDO

Consultas

En el caso de las consultas de acción, como INSERT, DELETE o UPDATE, el método exec devuelve el número de registros afectados.

```
$numRegistros = $miDB->exec('consultaSQLDeActualizacion’);
```

Si la consulta genera un conjunto de datos, como es el caso de SELECT, debes utilizar el método query, que devuelve un objeto de la clase PDOStatement.

```
$resultadoConsulta = $miDB->query('consultaSQLDeSeleccion');
```

PDO

Obtención y utilización de conjuntos de resultados

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
- 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.

PDO

Obtención y utilización de conjuntos de resultados

```
$resultadoConsulta = $miDB->query('consultaSQL’);
```

```
$registroArray = $resultadoConsulta->fetch();
```

Obtiene un registro en el array (asociativo y numérico por defecto) y avanza el puntero por el conjunto de registros

```
$registroObjeto = $resultadoConsulta->fetch(PDO::FETCH_OBJ);
```

Obtiene un registro en el objeto (asociativo) y avanza el puntero por el conjunto de registros Ambos métodos devuelven null cuando terminan de recorrer el conjunto de resultados.

PDO

Consultas preparadas. Forma 1

```
$consulta = $miDB->prepare('ConsultaPreparada VALUES (?,?,…));
```

Devuelve un objeto de la clase PDOStatement

Preparamos la consulta pendiente de dar valor a las ?

```
$consulta->bindParam(1, $var1);
```

```
$consulta->bindParam(2, $var2);
```

Sustituye las interrogaciones por el valor de las variables que pasamos en bind_param

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

PDO

Consultas preparadas

```
$consulta = $miDB->prepare('ConsultaPreparada VALUES (:param1, :param2,…));

$parametros = array(
        ":param1"=>"nombrecampo1",
        ":param2"=>"nombrecampo2");

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

PDO

Consultas preparadas

```
$consulta = $miDB->prepare('ConsultaPreparada VALUES (:param1, :param2,…));
```

Devuelve un objeto de la clase PDOStatement

Preparamos la consulta poniendo nombre a los campos precedidos de :

```
$consulta->bindParam(:param1, $var1);
```

```
$consulta->bindParam(:param2, $var2);
```

Sustituye las :param por el valor de las variables que pasamos en bind_param

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

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.

PDO

Transacciones

```
$miDB->beginTransaction(); //Deshabilita el modo autocommit
$resultadoConsulta1 = $miDB->exec('consultaSQL1’);
$resultadoConsulta2 = $miDB->exec('consultaSQL2’);
$miDB->commit(); //Confirma los cambios y los consolida
$miDB->rollback(); Revierte o deshace los cambios
```

PDO

Errores

Buscamos un comportamiento sólido y coherente en nuestra web

```
try { /Código susceptible de producir un error …
        throw new Exception ('MensajeDeError'); /Utilizamos la clase Exception …
} catch (Exception $miException) {
        echo'Error: '.$miException->getMessage();
        echo'Código de error: '.$miException->getCode();
}
```

PDO

Establecimiento de conexión

Tratamiento de errores

```
try {
    $conn = new PDO("mysql:host=".IP.";dbname=".BD, USUARIO, CLAVE);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 }
catch(PDOException $e)
    {$e->getMessage();}
finally {
    unset($conn); //Cerramos la conexion
}}
```

PDO

Errores

La clase PDO nos permite definir la fórmula que se utilizará cuento se produzca un error en el atributo PDO::ATTR_ERRMODE:

- PDO::ERRMODE_SILENT no se hace nada cuando hay error
- PDO::ERRMODE_WARNING genera un error de tipo E_WARNING
- PDO::ERRMODE_EXCEPTION lanza una excepción en PDOException

```
$miDB->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
        /Bloque de código que puede tener excepciones en el objeto PDO
} catch (PDOException $miExceptionPDO) {
        echo'Error: '.$miExceptionPDO->getMessage().'Código de error: '.$miExceptionPDO->getCode();
}
```