1 Curso POO PHP MySQLi

1.1 Sumario

1.2 MySQLi

Esta extensión desenvolveuse para aproveitar as vantaxes que ofrecen as versións 4.1.3 e posteriores de MySQL, e vén incluída con PHP a partir da versión 5. Ofrece un interface de programación dual, podendo accederse ás funcionalidades da extensión utilizando obxectos ou funcións de forma indiferente.

Por exemplo, para establecer unha conexión cun servidor MySQL e consultar a súa versión, podemos utilizar calquera das seguintes formas:

```
$conexion = new mysqli('localhost', 'usuario', 'contrasinal', 'base_de_datos');
print conexion->server_info;

$conexion = mysqli_connect('localhost', 'usuario', 'contrasinal', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

A utilización dos métodos e propiedades que achega a clase mysqli normalmente produce un código máis curto e lexible que se utilizas chamadas funcións. Entre as melloras que achega á antiga extensión mysql, figuran:

- Interface orientado a obxectos (nos dous exemplos anteriores, *$conexion* é un obxecto).
- Soporte para transaccións.
- Soporte para consultas preparadas.
- Mellores opcións de depuración.

1.2.1 Configuración de MySQLi

No ficheiro php.ini hai unha sección específica para as opcións de configuración propias de cada extensión. Entre as opcións que podes configurar para a extensión MySQLi están:

- **mysqli.allow_persistent**. Permite crear conexións persistentes.
- **mysqli.default_port**. Número de porto TCP predeterminado a utilizar cando se conecta ao servidor de base de datos.
- **mysqli.reconnect**. Indica se se debe volver conectar automaticamente en caso de que se perda a conexión.
- **mysqli.default_host**. Host predeterminado a usar cando se conecta ao servidor de base de datos.
- **mysqli.default_user**. Nome de usuario predeterminado a usar cando se conecta ao servidor de base de datos.
- **mysqli.default_pw**. Contrasinal predeterminado a usar cando se conecta ao servidor de base de datos.

1.2.2 Establecemento de conexións

Con MySQLi, establecer unha conexión co servidor significa crear unha instancia da **clase mysqli**. O construtor da clase pode recibir seis parámetros, todos opcionais, aínda que o máis habitual é utilizar os catro primeiros:

- o nome ou dirección IP do servidor MySQL ao que che queres conectar.
- un nome de usuario con permisos para establecer a conexión.
- o contrasinal do usuario.
- o nome da base de datos á que conectarse.
- o número do porto en que se executa o servidor MySQL.
- o socket ou a canalización con nome (named pipe) a usar.

Outra forma de obter unha instancia da clase mysqli é empregando a función **mysqli_connect**. Por tanto, establecer unha conexión á base de datos

"platega" pódese facer de calquera das seguintes formas:

```
$db = new mysqli('localhost', 'usuario', 'abc123.', 'platega');

$db = new mysqli();
$db->connect('localhost', 'usuario', 'abc123.', 'platega');

$db = mysqli_connect('localhost', 'usuario', 'abc123.', 'platega');
```

É importante verificar que a conexión se estableceu correctamente. Para comprobar o erro, en caso de que se produza, podes usar as seguintes propiedades (ou funcións equivalentes) da clase mysqli:

- **connect_errno** (ou a función **mysqli_connect_errno**) devolve o número de erro ou null se non se produce ningún erro.
- **connect_error** (ou a función **mysqli_connect_error**) devolve a mensaxe de erro ou null se non se produce ningún erro.

Por exemplo, o seguinte código comproba o establecemento dunha conexión coa base de datos "platega" e finaliza a execución se se produce algún erro:

```
@ $db = new mysqli('localhost', 'usuario', 'abc123.', 'platega');
$error = $db->connect_errno;
if ($error != null) {
echo "<p>Erro $error conectando á base de datos: $db->connect_error</p>";
exit();
}
```

Se unha vez establecida a conexión, queres cambiar a base de datos podes usar o método **select_db** (ou a función **mysqli_select_db** de forma equivalente) para indicar o nome da nova.

Unha vez finalizadas as tarefas coa base de datos, utiliza o método **close** (ou a función **mysqli_close**) para pechar a conexión coa base de datos e liberar os recursos que utiliza.

1.2.3 Execución de consultas

A forma máis inmediata de executar unha consulta con MySQLi é o método **query**, equivalente á función **mysqli_query**. Se se executa unha consulta de acción que non devolve datos (como unha sentenza SQL de tipo UPDATE, INSERT ou DELETE), a chamada devolve true se se executa correctamente ou false en caso contrario. O número de rexistros afectados pódese obter coa propiedade **affected_rows** (ou coa función **mysqli_affected_rows**).

No caso de executar unha sentenza SQL que si devolva datos (como un SELECT), estes devólvense en forma dun obxecto da **clase mysqli_result**.

```
@ $db = new mysqli('localhost', 'usuario', 'abc123.', 'platega');
$error = $db->connect_errno;
if ($error == null) {
   $resultado = $db->query('DELETE FROM stock WHERE unidades=0');
   if($resultado) print "<p>Borráronse $db->affected_rows rexistros.</p>";
}
$db->close();
```

O método query ten un parámetro opcional que afecta a como se obteñen internamente os resultados, pero non á forma de utilizalos posteriormente. Na opción por defecto, **MYSQLI_STORE_RESULT**, os resultados recupéranse todos xuntos da base de datos e almacénanse de forma local. Se cambiamos esta opción polo valor **MYSQLI_USE_RESULT**, os datos vanse recuperando do servidor segundo se vaian necesitando.

É importante ter en conta que os resultados obtidos se almacenarán en memoria mentres os esteas a usar. Cando xa non os necesites, pódelos liberar co método **free** da clase mysqli_result (ou coa función **mysqli_free_result**).

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

1.2.4 Transaccións

Se necesitas utilizar transaccións deberás asegurarte de que estean soportadas polo motor de almacenamento que xestiona as túas táboas en MySQL. Se utilizas InnoDB, por defecto cada consulta individual inclúese dentro da súa propia transacción. Podes xestionar este comportamento co método **autocommit** (función **mysqli_autocommit**).

```
// deshabilitamos o modo transaccional automático
```

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

Ao deshabilitar as transaccións automáticas, as vindeiras operacións sobre a base de datos iniciarán unha transacción que deberás finalizar utilizando:

- **commit** (ou a función **mysqli_commit**). Realizar unha operación "commit" da transacción actual, devolvendo true se se realizou correctamente ou false en caso contrario.
- **rollback** (ou a función **mysqli_rollback**). Realizar unha operación "rollback" da transacción actual, devolvendo true se se realizou correctamente ou false en caso contrario.

```
?
$db->query('DELETE FROM stock WHERE unidades=0');  // Inicia unha transacción
$db->query('UPDATE stock SET unidades=3 WHERE produto="XXXXXXX"');
?
$db->commit();  // Confirma os cambios
```

Unha vez finalizada cada transacción, a seguinte operación sobre a base de datos comezará outra nova transacción de forma automática.

1.2.5 Obtención e utilización de conxuntos de resultados

Xa sabes que ao executar unha consulta que devolve datos obtés un obxecto da clase **mysqli_result**. Esta clase segue os criterios de ofrecer un interface de programación dual, é dicir, unha función por cada método coa mesma funcionalidade que este.

Para traballar cos datos obtidos do servidor, tes varias posibilidades:

- **fetch_array** (función **mysqli_fetch_array**). Obtén un rexistro completo do conxunto de resultados e almacénao nun array. Por defecto o array contén tanto claves numéricas como asociativas. Por exemplo, para acceder ao primeiro campo devolto, podemos utilizar como clave o número 0 ou o seu nome indistintamente.

  ```
  $resultado = $db->query('SELECT produto, unidades FROM stock WHERE unidades<2');
  $stock = $resultado->fetch_array(); // Obtemos o primeiro rexistro
  $produto = $stock['produto']; // Ou tamén $stock[0];
  $unidades = $stock['unidades']; // Ou tamén $stock[1];
  print " < p>Produto $produto: $unidades unidades.</p>";
  ```

  Este comportamento por defecto pódese modificar utilizando un parámetro opcional, que pode tomar os seguintes valores:

  - ◊ **MYSQLI_NUM**. Devolve un array con claves numéricas.
  - ◊ **MYSQLI_ASSOC**. Devolve un array asociativo.
  - ◊ **MYSQLI_BOTH**. É o comportamento por defecto, no que devolve un array con claves numéricas e asociativas.

- **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 aos métodos anteriores, pero devolve un obxecto en lugar dun array. As propiedades do obxecto devolto correspóndense con cada un dos campos do rexistro.

Para percorrer todos os rexistros dun array, podes facer un bucle tendo en conta que calquera dos métodos ou funcións anteriores devolverá null cando non haxa máis rexistros no conxunto de resultados.

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

1.2.6 Consultas preparadas

Cada vez que se envía unha consulta ao servidor, este debe analizala antes de executala. Algunhas sentenzas SQL, como as que insiren valores nunha táboa, deben repetirse de forma habitual nun programa. Para acelerar este proceso, MySQL admite consultas preparadas. Estas consultas almacénanse no servidor listas para ser executadas cando sexa necesario.

Para traballar con consultas preparadas coa extensión MySQLi de PHP, debes utilizar a **clase mysqli_stmt**. Utilizando o método **stmt_init** da clase mysqli (ou a función **mysqli_stmt_init**) obtés un obxecto da devandita clase.

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

Os pasos que debes seguir para executar unha consulta preparada son:

- Preparar a consulta no servidor MySQL utilizando o método **prepare** (función **mysqli_stmt_prepare**).
- Executar a consulta, tantas veces como sexa necesario, co método **execute** (función **mysqli_stmt_execute**).
- Unha vez que xa non se necesita máis, débese executar o método **close** (función **mysqli_stmt_close**).

Por exemplo:

```
$consulta = $db->stmt_init();
$consulta->prepare('INSERT INTO cursos (cod, nome) VALUES ("G1301006", "POO para plataformas web con PHP e MySQL")');
$consulta->execute();
$consulta->close();
$db->close();
```

O problema é que de pouco serve preparar unha consulta de inserción de datos como a anterior, se os valores que insire son sempre os mesmos. Por este motivo as consultas preparadas admiten parámetros. Para preparar unha consulta con parámetros, en lugar de poñer os valores debes indicar cun signo de interrogación a súa posición dentro da sentenza SQL.

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

E antes de executar a consulta tes que utilizar o método **bind_param** (ou a función **mysqli_stmt_bind_param**) para substituír cada parámetro polo seu valor. O primeiro parámetro do método bind_param é unha cadea de texto na que cada carácter indica o tipo dun parámetro, segundo a seguinte táboa.

Caracter	**Tipo do parámetro**
I	número entero
D	número real (doble precisión)
S	cadea de texto
B	contido en formato binario (BLOB)

Isto é:

```
$consulta = $db->stmt_init();
$consulta->prepare('INSERT INTO cursos (cod, nome) VALUES (?, ?)');
$cod_produto = "G1301006";
$nome_produto = "POO para plataformas web con PHP e MySQL";
$consulta->bind_param('ss', $cod_produto, $nome_produto);
$consulta->execute();
$consulta->close();
$db->close();
```

No caso das consultas que devolven valores, pódese utilizar o método **bind_result** (función **mysqli_stmt_bind_result**) para asignar a variables os campos que se obteñen tras a execución. Utilizando o método **fetch** (**mysqli_stmt_fetch**) percórrense os rexistros devoltos.

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

--Víctor Lourido 19:10 15 jul 2013 (CEST)