Traballar con BD en PHP

Sumario

Acceso a Bases de Datos

Poderemos desde PHP conectarnos a diferentes bases de datos. Neste curso centrarémonos no acceso á MySQL.

Creación de Bases de Datos

Podemos crear a base de datos desde un script ou ben desde MySQL Workbench ou phpMyAdmin.

Conexión a Bases de Datos. *mysqli_connect() - mysqli_close()*

Cada xestor de Base de datos (XBD) ten as súas funcións de conexión en PHP (PHP tamén ten un método xeral PDO que veremos máis adiante). En xeral, para conectarnos a unha base de datos teremos que coñecer:

- Servidor de bases de datos
- Usuario
- Contrasinal
- Nome da base de datos a empregar

Para MySQL existen 2 versións, a recomendable é a "i" (improved, mellorada):

~~**mysql_connect** (~~ servidor, usuario, passwd,...) (versión PHP<5.5.0)

mysqli_connect (servidor, usuario, passwd, baseDatos, ...) (versión PHP>5.5.0)

A sentencia mysqli_connect é un alias de mysql::_construct (ver https://www.php.net), e devolve un identificador/obxecto que representa unha conexión de base de datos MySQL ou ***False*** se hai un erro. Vexamos un exemplo en PHP, se temos definida a base de datos "proba" no servidor "db", con usuario "usuario" e contrasinal "abc123.", para conectarnos á base datos podemos facer algo así:

```
<!DOCTYPE html>
<html>
<head>
   <title>Conexión a bases de datos</title>
</head>
<body>
<?php
// Créase a conexión ao servidor de mysql.
// A sentencia die() detén a execución da páxina
// devolvendo o último erro do servidor de mysql.

$conexion=mysqli_connect("db","usuario","abc123.","proba");
```

```
if ($conexion) {
    echo "Feita a conexión coa base de datos.<br>";
} else {
   echo "Erro conectando coa base de datos";
}
// Pechamos a conexion.
mysqli_close($conexion);
?>
</body>
</html>
```

Unha vez feita a conexión coa base de datos, poderemos enviar consultas á base de datos, coa función mysqli_query.

Consultas á base de datos. *mysqli-query()*

A función mysqli_query() envía unha sentencia SQL ao servidor para que este a execute. Será:

mysqli_query(conexión,consulta);

A consulta poderá ser calquera consulta SQL válida, podendo así crear, modificar, e eliminar táboas ou índices, e inserir, actualizar, ou consultar os datos da BD sempre que teñamos os permisos correspondentes.

OLLO: Valores **devoltos** de ***mysqli_query()***

- Para SELECT, SHOW, DESCRIBE, EXPLAIN... retornan un **conxunto de resultados** ou FALSE
- Para INSERT, UPDATE, DELETE, DROP, ... devolve TRUE ou FALSE (foi correcto, ou non)

Unha vez realizada a consulta teremos que acceder ao conxunto de resultados.
Para iso pódense utilizar varias funcións, entre outras, ***mysqli_result(), mysqli_fetch_row(), mysqli_fetch_array(), mysqli_num_rows(), mysqli_affected_rows().***
Principalmente empregaremos **mysqli_fetch_array():**
Esta función devolve un array que corresponde a unha das filas devoltas. Permite tanto acceder aos valores como un array asociativo polos nomes dos campos ou con índices numéricos. Emprégase habitualmente un bucle para percorrer ese array:

```
<!DOCTYPE html>
<html>
<head>
   <title>Conexión a bases de datos</title>
      <meta charset="UTF8">
</head>
<body>
```

```
<?php
    //$conexion=mysqli_connect("localhost","usuario","Password", "proba");   //Co xampp
      $conexion=mysqli_connect("dbXdebug","usuario","abc123.", "proba");  //Co docker de
                                                                    // clase
   if ($conexion) {
      mysqli_set_charset($conexion,"utf8");
      $resultado=mysqli_query($conexion,"SELECT codCliente,nome,apelidos from cliente");
      if ($resultado != FALSE) {
         while($fila=mysqli_fetch_array($resultado))
            echo $fila["codCliente"]," ",$fila["nome"]," ", $fila["apelidos"],"<br>";
         }
   }  else {
         echo "Fallou a conexión coa base de datos";
   }
   mysqli_close($conexion);      // Pechamos a conexion.
?>
</body>
</html>
```

Traballando con imaxes

Subida de ficheiros ao servidor

Para almacenar ficheiros no servidor, podemos subir o ficheiro e almacenalo nunha carpeta determinada no servidor. Teremos primeiro que ter un formulario que envíe os datos en "modo ficheiro". Isto faise enviando por **POST** e engadindo ao form o atributo **enctype="multipart/form-data"**:

```
<form method="POST" action="recibeFich.php" enctype="multipart/form-data">
 Ficheiro:<input name="meuArquivo" type="file"/>
    <input name="enviar" type="submit" value="Enviar"/>
<form>
```

OLLO: As variables post_max_size, upload_max_filesize do php.ini limitarán o tamaño do ficheiro.

Teremos un script que xestione a recepción do ficheiro. En PHP recoméndase empregar o array asociativo $FILES para ler os datos dos ficheiros subidos por POST. Este array ten os seguintes índices:

- ✔ **$_FILES['meuArquivo']['name']:** o nome orixinal do ficheiro.
- ✔ **$_FILES['meuArquivo']['type']:** o tipo MIME do ficheiro, image/gif, application/pdf, application/msword,... etc
- ✔ **$_FILES['meuArquivo']['size']:** O tamaño do arquivo en bytes.
- ✔ **$_FILES['meuArquivo']['tmp_name']:** A ubicación do arquivo temporal que se crea cando se sube un ficheiro ao servidor. Nesta variable están os datos, pero se non son copiados ou movidos, pódense perder pois PHP elimina este ficheiro temporal ao cabo dun tempo.

Se estamos gardando as nosas imaxes nunha carpeta **imaxes** o ficheiro recibeFich.php podería ser:

```
if(isset($_POST["enviar"])){
       $tmp_name = $_FILES['meuArquivo']['tmp_name'];
         //SE O FICHEIRO ESTÁ SUBIDO POR POST:
      if (is_uploaded_file($tmp_name))
      {
        $img_file = $_FILES['meuArquivo']['name'];  //O NOME
        $img_type = $_FILES['meuArquivo']['type'];  // A EXTENSIÓN
         // SE É UNHA IMAXE:
        if (((strpos($img_type, "gif") || strpos($img_type, "jpeg") ||
                strpos($img_type, "jpg")) || strpos($img_type, "png")))
        {
          //COMPROBAMOS QUE PODEMOS ESCRIBIR NA CARPETA IMAXES E TODO FOI BEN:

         if (move_uploaded_file($tmp_name, "imaxes/". $img_file))
            {
                echo "arquivo subido con éxito";            } }
      }}
```

Almacenamento de imaxes na bases de datos

Aínda que menos frecuente, para almacenar as imaxes nunha base de datos, podemos definir un campo BLOB, que permite almacenar datos binarios. En Mysql existen os seguinte tipos BLOB:

```
TINYBLOB: Ata 255 bytes
BLOB: Ata 65 Kb
MEDIUMBLOB: Ata 16 Mb
LONGBLOB: Ata 4 Gb
```

*OLLO: As variables post_max_size, upload_max_filesize do **php.ini** limitarán o tamaño do ficheiro.*

Por exemplo, podemos crear unha táboa imaxe:

```
CREATE TABLE `imaxe` (
  `imaxe` mediumblob,
 `nome_imaxe` varchar(30),
 `tipo_imaxe` varchar(30)
)
```

Poderíamos introducir valores directamente desde phpMyAdmin, ou introducir datos desde o navegador cliente. Podemos ter de novo un formulario para subir ficheiros:

```
<form method="POST" action="gardaFich.php" enctype="multipart/form-data">
 Imaxe:<input name="meuArquivo" type="file"/>
    <input name="enviarFich" type="submit" value="Enviar"/>
<form>
```

e recoller o enviado e inserir un rexistro na nosa táboa "imaxes" deste xeito:

gardaFich.php

```
<?php
// Conexion á base de datos
$conex=mysqli_connect("db", "root", "root","proba") or die(mysqli_error());

// COMPROBAMOS ERROS...
if (!isset($_FILES["meuArquivo"]) || $_FILES["meuArquivo"]["error"] > 0)
{
    echo "Houbo un erro.";
}
else
{
   mysqli_set_charset($conex,"UTF8");
    // VERIFICAMOS QUE O TIPO DA IMAXE É COÑECIDO E O TAMAÑO NON SUPERA OS 16MB
    $permitidos = array("image/jpg", "image/jpeg", "image/gif", "image/png");
    $limite = 16*1024*1024;

    if (in_array($_FILES['meuArquivo']['type'], $permitidos) && $_FILES['meuArquivo']['size'] <=
$limite)
    {
       $imaxe_temporal = $_FILES['meuArquivo']['tmp_name'];
      $tipo = $_FILES['meuArquivo']['type'];
   $nomeG=$_FILES['nomeArquivo']['name'];
```

```
     //LEMOS O CONTIDO DO FICHEIRO E ESCAPAMOS OS CARACTERES PARA QUE SE ALMACENEN
   //CORRECTAMENTE NA BASE DE DATOS

         $data=file_get_contents($imaxe_temporal);
         $data = mysqli_real_escape_string($conex,$data);

        //INSERIMOS NA BASE DE DATOS
        $resultado = mysqli_query($conex,"INSERT INTO imaxe  VALUES ('$data','$nomeG',
'$tipo')") ;

        if ($resultado)    {
            echo "Arquivo copiado correctamente";
        }
        else        {
            echo "Houbo un erro:";
          printf("Error: %s\n", mysqli_error($conex));
        }
    }
    else     {
        echo "FORMATO NON PERMITIDO OU TAMAÑO MOI GRANDE.";
    }
}
?>
```

Para mostrar as imaxes da base de datos, podemos entre outras opcións que empregar a función ***base64_encode($datosBinarios)***, que codifica os datos binarios coa codificación base64. Logo indicarémoslle a img que o seu src será unha imaxe jpg (ou no formato que teñamos) codificada deste xeito. Algo así:

```
<?php
// Conexion á base de datos
$conex=mysqli_connect("localhost", "proba", "abc123.","proba") or die(mysqli_error($conex));
      mysqli_set_charset($conex,"UTF8");
    // CONSULTAMOS A BASE DE DATOS.
    $consulta = "SELECT imaxe, tipo_imaxe FROM imaxe WHERE nome_imaxe='$nome' ";
   $resultado = mysqli_query($conex, $consulta) or die(mysqli_error($conex));


   if (mysqli_num_rows($resultado)>0)
   {
   while($fila=mysqli_fetch_assoc($resultado))     {
      $imaxe = $fila['imaxe']; // Datos binarios da imaxe.
      $tipo = $fila['tipo_imaxe'];  // Mime Type da imaxe.

   echo '<img src="'.base64_encode($imaxe).'"/>';
   }
   }

?>
```

Sentencias preparadas

OLLO: Para as sentencias preparadas é conveniente conectarnos á BD co estilo orientado a **obxectos**.

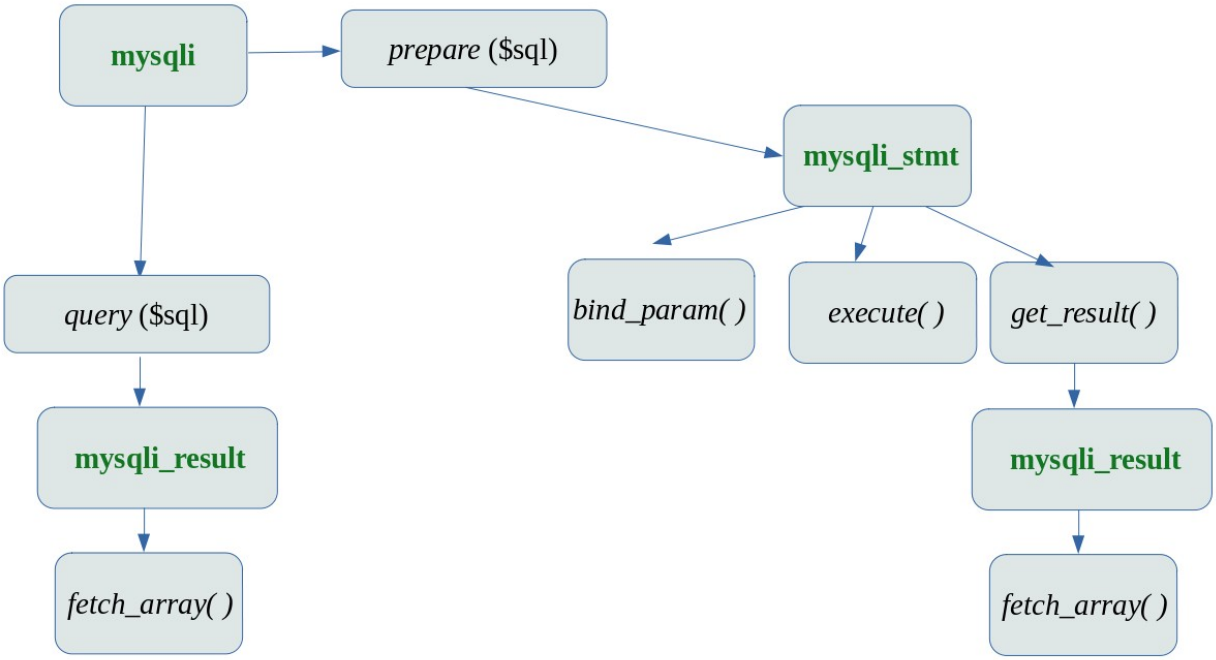
En PHP emprégase a palabra reservada **new** para crear un obxecto dunha clase:

```
$conexion =  new mysqli(...);
```

E accedemos aos métodos e propiedades empregando o operador frecha **->**

```
echo $conexion->connect_error;
```

A clase mysqli:



Unha **sentencia preparada** (*sentencia parametrizada*) úsase para executar a mesma sentencia repetidamente e con gran rendemento (funcionan como plantillas compiladas para SQL) . Provocan ademais unha mellora de seguridade da BD (por exemplo fronte a Inxección de SQL).

A sentencia só necesita ser analizada unha vez (ou preparada) e executada logo unha ou varias veces con diferentes parámetros.

As sentencias preparadas empregan 2 pasos:

- ✗ Preparación (prepare)
- ✗ Unir parámetros e Execute (Execución)

Preparación

Creamos unha plantilla da sentencia SQL e envíase á base de datos. Algúns valores estarán sen especificar , serán os parámetros representados por un interrogante:

```
$conexion->prepare("INSERT INTO Cliente   VALUES (?, ?, ?)");
```

Despois, a base de datos analizará esa consulta, compilaraa, e realizará a optimización da mesma sobre a sentencia SQL, gardando o resultado sen executalo, para un uso posterior.

Unir parámetros e execución

Debemos enlazar valores cos parámetros (con ***bind_param()***), e executar a sentencia (con ***execute()***). Podemos executar a sentencia tantas veces como se queira, con valores diferentes. Fíxate que agora os parámetros non precisan estar entre comiñas.

Entenderemos mellor isto cun exemplo, seguindo cos clientes:

```
$servidor="db";
$usuario="root";
$passwd="root";
$base="proba";

//CONECTAMOS
$conexion = new mysqli($servidor, $usuario, $passwd, $base);   //CONECTAMOS COA NOTACIÓN POO
if($conexion->connect_error)
   die("Non é posible conectar coa BD: ". $conexion->connect_error);
$conexion->set_charset("utf8");
//PREPARAMOS A SENTENCIA:
$sentenciaPrep=$conexion->prepare("INSERT INTO cliente (codCliente,nome,apelidos)
VALUES(?, ?, ?)");

// DAMOS VALORES AOS PARÁMETROS E EXECUTAMOS:
$codCliente=100;
$nome="Xan";
$apelidos="Fieito";
$sentenciaPrep->bind_param('iss',$codCliente, $nome, $apelidos);   //INDICAMOS O TIPO DAS
VARIABLES

if(!$sentenciaPrep->execute() )  //EXECUTAMOS A CONSULTA
   echo "Houbo un erro na execución da consulta";

$codCliente=101;
$nome="Eva";
$apelidos="Loureiro";
$sentenciaPrep->bind_param('iss',$codCliente, $nome, $apelidos);

if(!$sentenciaPrep->execute() )
   echo "Houbo un erro na execución da consulta";

$sentenciaPrep->close();   //PECHAMOS AS CONEXIÓNS
$conexión->close();
```

No método ***bind_param()*** especificamos á base de datos que parámetros terá a consulta e de que tipo son. Os argumentos poden ser de catro tipos:

i: integer

s: string

d: double

b: blob

Temos que especificar un por cada parámetro!

Exemplo con SELECT para obter datos empregando sentencias preparadas:

```
$sentencia=$conexion->prepare("SELECT * FROM cliente where nome = ?");
$nome="Xan";    //OU $_GET['nome'] SE VIMOS DUN FORMULARIO
$sentencia->bind_param("s",$nome);
$sentencia->execute();
$resultado=$sentencia->get_result();

while($fila=$resultado->fetch_array(MYSQLI_BOTH) )
    echo $fila['codCliente']." ".$fila['nome']." ".$fila['apelidos'];

$sentencia->close();
$conexion->close();
```

Para saber máis: https://www.php.net/manual/es/pdo.prepared-statements.php

Exemplo resolto: https://manuais.iessanclemente.net/index.php/Exemplo_empregando_as_clase_mysqli_e_mysqli_stmt

Exemplo de conexión con estilo orientado a obxectos:

Para traballar co estilo orientado a obxectos existe unha clase definida en php, mysqli:

https://www.php.net/manual/es/class.mysqli.php

que representa a conexión entre php e a base de datos. Os métodos definidos para esta clase son os que levamos usando ata agora sen utilizar o **mysqli_** previo, e sen utilizar a variable que representa a conexión. A conexión xa está representada no propio obxecto que ten os métodos:

Por exemplo,

Estilo procedemental	**Estilo orientado a obxectos**
$con=mysqli_connect(...)	$mysqli = new mysqli(...)
mysqli_query($conexion, "SELECT ...");	mysqli->query("SELECT ...");

```
<!DOCTYPE html>
<html>
<head>
   <title>Conexión a bases de datos</title>
      <meta charset="UTF8">
</head>
<body>
<?php
    //AGORA CREAMOS UN OBXECTO DA CLASE mysqli

   // $mysqli= new mysqli("localhost","usuario","Password", "proba");   //En xampp
  $mysqli=new mysqli('db','root','root', 'proba');    //Se estou co docker de clase
  if ($mysqli->connect_error) {
    die('Erro de Conexión (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
  else
   {
      $mysqli->set_charset("utf8");
   $resultado=$mysqli->query("SELECT codCliente,nome,apelidos from cliente");
   if ($resultado != FALSE)
         {
         while($fila=$resultado->fetch_array())
            echo $fila["codCliente"]," ",$fila["nome"]," ",$fila["apelidos"],"<br>";
         }
   }

   $mysqli->close();    // Pechamos a conexion.
?>
</body>
</html>
```

Acceso a base de datos con PDO (PHP Data Objects)

Podes aprender máis en:

https://www.php.net/manual/es/book.pdo.php

https://manuais.iessanclemente.net/index.php/Acceso_a_bases_de_datos_con_PDO

A extensión PDO (PHP Data Objects) define unha interfaz para poder acceder a diferentes bases de Datos en PHP. Cada base de datos ten definida esa interfaz PDO para poder ser empregada.
A extensión PDO de php non é suficiente para realizar as funcións de base de datos: temos que empregar un controlador de PDO específico de cada base de datos, para poder acceder a ese servidor. Deste modo, PDO proporciona unha capa de abstracción coa BD, de modo que se empregan as mesmas funcións para conectarse e realizar consultas, polo que a nosa aplicación permite o cambio de SXBD sen dificultades.
Por exemplo temos os seguintes controladores:

- PDO_SQLSRV
- PDO_MYSQL
- PDO_DLIB
- ...

Para saber os controladores PDO no noso sistema:

```
print_r(PDO::getAvailableDrivers());
```

Conexión coa base de datos mysql

Crearemos un obxecto PDO:

```
$dsn='mysql:dbname=testdb;host=127.0.0.1';
$user='dbuser';
$password='dbpass';
$dbh=new PDO($dsn,$user,$password);
```

//EXEMPLO CO DOCKER DE CLASE

```
$servidor="db";
$usuario="root";
$passwd="root";
$base="proba";
//CONECTAMOS
$pdo = new PDO("mysql:host=$servidor;dbname=$base;charset=utf8mb4", $usuario, $passwd);
```

E agora teríamos o noso obxecto PDO para enviar as consultas.

PDO e excepcións

PDO permite empregar as excepcións para xestionar os erros, podemos por tanto empregar os bloques ***try...catch*** para xestionar estes erros:

```
$servidor="db";
$usuario="root";
$passwd="root";
$base="proba";
try {
//CONECTAMOS
$pdo = new PDO("mysql:host=$servidor;dbname=$base;charset=utf8mb4", $usuario, $passwd);
   //Para xerar excepcións cando se informe dun erro
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   echo "Todo ben na conexión";
} catch (Exception $e) {
   echo "Erro ao conectar co servidor MySQL: ". $e->getMessage();
}
finally {
   $pdo = null;   //PECHAMOS A CONEXIÓN COA BASE DE DATOS
}
```

PDO pode xestionar os erros de 3 xeitos distintos, que podemos indicar co método setAttribute:

- ✗ **PDO::ERRMODE_SILENT**: é o modo por defecto. Temos que xestionar os erros con ->errorCode( ) e ->errorInfo( ).
- ✗ **PDO::ERRMODE_WARNING**: xera erros warning PHP e a execución continúa.
- ✗ **PDO::ERRMODE_EXCEPTION**: produce unha excepción que podemos xestionar

No exemplo anterior xestionamos o erro da conexión, tamén poderíamos xestionar o erro enviando unha consulta:

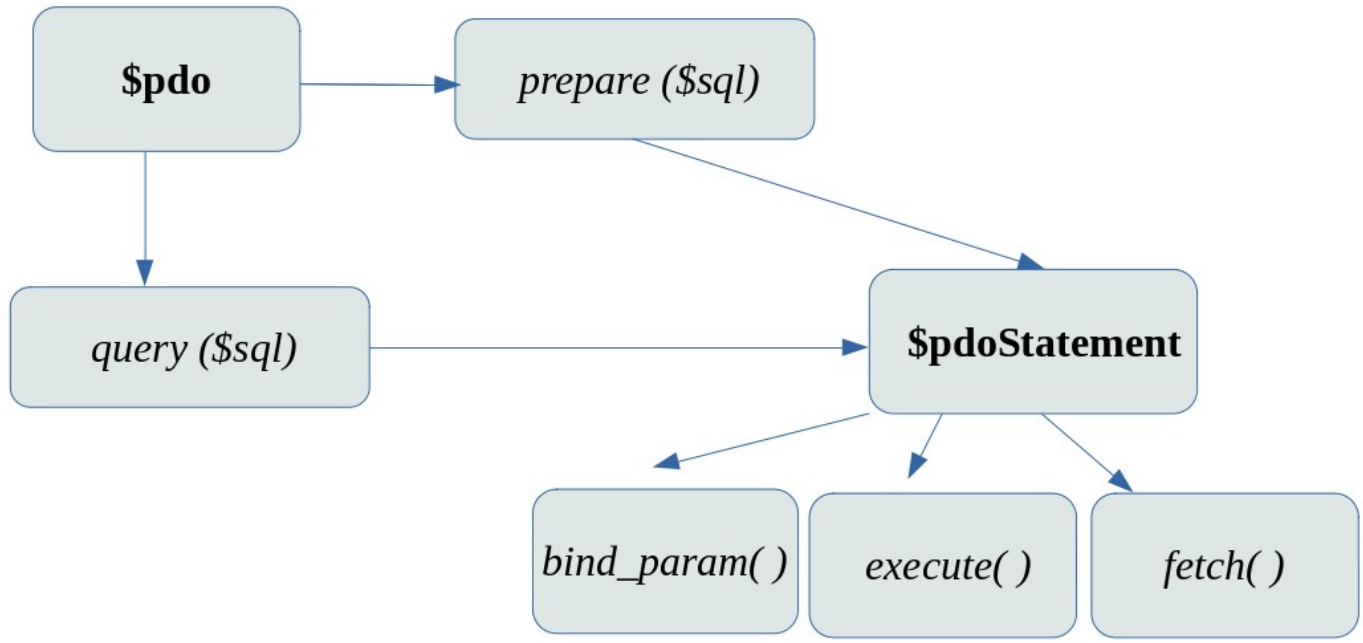
```
try{
   $pdoStatement=$pdo->query("selects * from cliente");
}
catch(PDOException $e) {
   echo "erro na consulta!";
}
```

Consultas preparadas con PDO

Cando a consulta ten algún parámetro é conveniente empregar as consultas ***preparadas*** ou ***parametrizadas***, do mesmo xeito que xa vimos coa clase **mysqli**. As sentencias preparadas empregan 2 pasos:

- ✗ Preparación (prepare)
- ✗ Unir parámetros e Execute (Execución)

O esquema das clases agora é semellante ao xa visto en **mysqli** (***query( )*** agora devolve un obxecto da clase **pdoStatement**):



Por exemplo, coa nosa táboa de clientes:

....

```
try { //CONECTAMOS
$pdo = new PDO("mysql:host=$servidor;dbname=$base;charset=utf8", $usuario, $passwd);
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
//PREPARAMOS A SENTENCIA:

$pdoStatement=$pdo->prepare("SELECT * FROM cliente ");
$pdoStatement->execute();
   // SE NON TEMOS PARÁMETROS PODERÍAMOS RESUMIR AS 2 LIÑAS ANTERIORES EN:
   //  $pdoStatement=$pdo->query("SELECT * FROM cliente ");
echo "<table><tr><th>Nome</th><th>Apelidos</th></tr>";
while($fila=$pdoStatement->fetch(PDO::FETCH_ASSOC) )
   echo "<tr><td>".$fila['nome']." </td><td>".$fila['apelido']."</td></tr>";
echo "<table>";
}
catch(PDOException $e) {
echo "Erro ao conectar co servidor MySQL: ".$e->getMessage();
}
$pdo=null;

?>
```

Marcadores en consultas preparadas

bindParam() : recibirá uns parámetros ou marcadores que poden ser **anónimos** ou **coñecidos**

Con marcadores ***anónimos*** (co símbolo ?):

```
try { //CONECTAMOS
$pdo = new PDO("mysql:host=$servidor;dbname=$base;charset=utf8", $usuario, $passwd);
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
//PREPARAMOS A SENTENCIA:

$pdoStatement=$pdo->prepare("INSERT INTO cliente (codCliente, nome, apelido) VALUES (?,?,?)");
$id=101;
$nome="Xiao";
$apelido="Ferreiro";
$pdoStatement->bindParam(1, $id);
$pdoStatement->bindParam(2, $nome);
$pdoStatement->bindParam(3, $apelido);
$pdoStatement->execute();

// OU SEN EMPREGAR bindParam , asignando valores empregando un array:
$pdoStatement->execute(array(102,"Ana","Sánchez"));

}
```

Con marcadores **coñecidos**. Agora indicaremos o nome das variables:

```
/PREPARAMOS A SENTENCIA:
$pdoStatement=$pdo->prepare("INSERT INTO cliente (codCliente, nome, apelido)
   VALUES (:codCliente,:nome,:apelido)");
$id=106;
$nome="Xan";
$apelido="Ferrán";
$pdoStatement->bindParam(':codCliente', $id);
$pdoStatement->bindParam(':nome', $nome);
$pdoStatement->bindParam(':apelido', $apelido);
$pdoStatement->execute();

   //Tamén se pode crear un array asociativo cos marcadores, que será o argumento de execute( ):
   $datosCliente= array('codCliente'=>108,'nome'=>'Xose', 'apelido'=>'Gómez');
   $pdoStatement->execute($datosCliente);
```

Con ***bindParam()*** tamén poderíamos indicar as características do parámetro:

```
   $pdoStatement->bindParam(':codCliente',$codCliente, PDO::PARAM_INT);
   $pdoStatement->bindParam(':nome',$nome, PDO::PARAM_STR, 45);
   $pdoStatement->bindParam(':nome',$apelido, PDO::PARAM_STR, 45);
```

Tamén poderíamos enlazar valores con ***bindValue( )***

Recollendo os resultados

Se traballamos con consultas de selección, o obxecto ***PDOStatement*** resultado do execute() gardará os datos. Podemos acceder a ela, empregando o método ***fetch()*** ou ***fetchAll( ).*** Por exemplo:

```
... //INICIAMOS AS VARIABLES PARA A CONEXIÓN
try { //CONECTAMOS
$pdo = new PDO("mysql:host=$servidor;dbname=$base;charset=utf8", $usuario, $passwd);
$pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

$pdoStatement=$pdo->prepare("SELECT * FROM cliente ");
$pdoStatement->execute();

echo "<table><tr><th>Nome</th><th>Apelidos</th></tr>";
while($fila=$pdoStatement->fetch(PDO::FETCH_ASSOC) )
   echo "<tr><td>".$fila['nome']." </td><td>".$fila['apelido']."</td></tr>";

echo "<table>";
}
catch(PDOException $e) {
echo "Erro ao conectar co servidor MySQL: ".$e->getMessage();
}
$pdo=null;
?>
```

O método fetch( ) obtén a seguinte fila do conxunto de resultados, e permite recibir un parámetro indicando como queremos que sexa a fila que devolve. Entre outros:

- **PDO::FETCH_ASSOC:** devolve un array indexado cos nomes das columnas do conxunto de resultados
- **PDO::FETCH_BOTH** (predeterminado): devolve un array indexado e asociativo, indexado desde 0 por orden do conxunto de resultado, e asociativo co nome das columnas do conxunto de resultados.
- **PDO::FETCH_OBJ:** devolve un obxecto anónimo con nomes das propiedades que se corresponden cos nomes das columnas do conxunto de resultados.

Cifrado de contrasinais na Base de datos con PHP

Os contrasinais dos usuarios na base de datos deberanse gardar cifrados, para que calquera acceso indebido á mesma non proporcione todos os contrasinais dos nosos usuarios: gardaremos en vez do contrasinal en texto plano o resultado de pasarlle unha función de hash a dito contrasinal.
Despois, cando comprobemos o contrasinal tecleado teremos que comprobar que o resultado de pasar a función de hash ao contrasinal tecleado é o mesmo que o gardado na base de datos.
As funcións básicas/antigas de hashing en php eran as seguintes (pasámoslle un string e nos devolve o string hasheado):

- **md5**

```
string md5 (string $str [, bool $raw_output = false ])

Exemplo:  $contrasinalHashMd5= md5($contrasinalTecleado);
```

Calcula un hash co algoritmo md5. Se se establece _$raw*output* como true devolverá un raw binario cunha lonxitude de 16. Por defecto un hash de 32 caracteres hexadecimal.

- **sha1**

```
string sha1 (string $str [, bool $raw_output = false ])
```

Calcula un hash con el algoritmo sha1. Se se establece _$raw*output* como true devolverá un raw binario con una longitud de 20. Por defecto un hash de 40 caracteres hexadecimal.

- **hash**

```
string hash ( string $algoritmo, string $data [, bool $raw_output = false ] )
```

A función toma primeiro o algoritmo que se desexa empregar, *$algoritmo*, y despois o string que se desexa encriptar, *$data*. O algoritmo puede ser **md5, sha128, sha256**... Devolverá o contrasinal encriptado.

A avantaxe de empregar estas funcións é que podemos introducir directamente os contrasinais desde PHPMyAdmin. A desvantaxe é que NON son seguras 100%.

Librería Hash de contrasinais

A extensión hash para contrasinais de PHP crea un password complexo, que se axusta aos estándares de seguridade do momento, polo que é o **método máis recomendado**.

Podemos empregar a función ***password_hash***, co contrasinal que queremos "hashear", e a extensión xa o fai directamente. Recoméndase gardar o resultado nun campo de 255 caracteres.
Empregará o algoritmo que lle indiquemos, se indicamos PASSWORD_DEFAULT empregará o algoritmo máis forte. Ver https://www.php.net/manual/es/function.password-hash

password_hash

string password_hash (string $password, integer $algoritmo [,array $options])

Por exemplo, podemos "hashear" o noso contrasinal con:

```
   $hasheado = password_hash("abc123.", PASSWORD_DEFAULT, [15]);
```

Dependendo do número ($options) o algoritmo será máis complexo e tardará máis en xerarse o hash. Pódese considerar como o número de veces que o algoritmo "hashea" o contrasinal

Para comprobar o contrasinal gardado, agora temos que empregar a función

boolean password_verify($password, $ hash)

Devolverá TRUE se o contrasinal coincide co almacenado, e FALSE en caso contrario.
Exemplo:

```
... XA FIXEMOS A CONSULTA E TEMOS O PASSWORD GARDADO NA BD NA VARIABLE $hash:

if(password_verify($passwordTecleado, $hash)) {
     //PASSWORD CORRECTO    }
else {
   //PASSWORD INCORRECTO    }
```

Empregando esta extensión de encriptación de PHP, a nosa aplicación estará nos últimos estándares de seguridade (hai algúns anos SHA-1 era o mellor). A extensión vaise adaptando aos cambios e a nosa aplicación será segura.