Consultas

Existen diferentes formas de generar consultas desde MySQL, pero la más usada es el método query().

El método query($cad) de la clase PDO recibe una instrucción SQL válida. Devuelve FALSE si hubo algún error o un objeto PDOStatement si se ejecutó con éxito.

Si se trata de una consulta, es posible recorrer las filas devueltas con un foreach. En cada iteración del bucle se tendrá una fila, representada como un array en que las claves son los nombres que aparecen en la cláusula SELECT.

Para ejecutar la consulta SELECT, si no tenemos parámetros en la consulta, podremos usar ->**query**() del objeto PDO.

Ejemplo:

```php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';

try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    echo "Conexión realizada con éxito<br>";
    $sql = 'SELECT nombre, clave, rol FROM usuarios';
    $usuarios = $bd->query($sql);
    echo "Número de usuarios: " . $usuarios->rowCount() . "<br>";
    foreach ($usuarios as $usu) {
        print "Nombre : " . $usu['nombre']."<br>";
        print " Clave : " . $usu['clave'] . "<br>";
    }

    } catch (PDOException $e) {
        echo 'Error con la base de datos: ' . $e->getMessage();
    }
```

Consultas preparadas

Una consulta preparada es una sentencia SQL precompilada, que se puede ejecutar múltiples veces simplemente enviando datos al servidor. La sentencia puede ser la misma o similar.

Las consultas preparadas se usan como plantillas que podemos rellenar, más adelante, con valores reales que previamente son obtenidos desde formularios.

Con las consultas preparadas se mejora la seguridad y el rendimiento de la aplicación. Por ejemplo, nos ayudará a evitar la inyección SQL.

La consulta preparada es un objeto de tipo PDOStatement.

Estas consultas se inicializan una sola vez con el método prepare() y se ejecutan, las veces que sea necesario, con el método execute() asignando diferentes valores a los parámetros.

Las sentencias preparadas se realizan en 3 pasos:

1. **Preparación:** Se envía una plantilla de la consulta al servidor, quien revisa la sintaxis y prepara los recursos necesarios. Algunos valores, llamados parámetros, se dejan sin especificar.

2. **Vinculación:** Se revisan los tipos de datos de cada parámetro. La base de datos analiza, compila y realiza la optimización de la consulta sobre la sentencia SQL, y guarda el resultado sin ejecutarlo.

3. **Ejecución:** Sólo cuando le decimos que se ejecute, la aplicación enlaza los valores que nosotros enviamos con los parámetros**,** y la base de datos ejecuta la sentencia. La misma sentencia se puede ejecutar tantas veces como queramos con valores diferentes.

Ventajas de las consultas preparadas:

- Se ahorra tiempo de procesamiento, puesto que la preparación de la consulta se realiza sólo una vez. Después se envían los parámetros y se ejecuta las veces que queramos.

- Como sólo enviamos al servidor los parámetros de la consulta se reduce el ancho de banda consumido.

- Los riesgos de sufrir ataques de inyección SQL se reducen. Los valores de los parámetros se trasmiten usando un protocolo diferente.

Para **construir una sentencia preparada** hay que incluir **marcadores** en nuestra sentencia SQL.

Existen dos tipos de marcadores: **anónimos y conocidos**.

La elección de usar marcadores anónimos o conocidos afectará a cómo se asignan los datos a esos marcadores.

a. Ejemplo de cómo construir una **consulta preparada con marcadores anónimos** o de **posición**.

   Cuando preparamos la consulta usamos el símbolo "?" para indicar un parámetro. Al ejecutar se asocian por orden los símbolos de interrogación con los valores del array que se pasa como argumento a execute().

```
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol)
                     values (?, ?, ?)");
```

b. Ejemplo de cómo construir una consulta preparada con **marcadores conocidos o** parámetros **por nombre**. Cuando preparamos la consulta usamos nombres para los parámetros seguido de símbolo ":". En este caso se usarán esos nombres como clave del array de execute().

```
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol)
              VALUES (:nombre, :clave, :rol)");
```

Recuerda que **siempre** debemos **usar marcadores**. No hacerlo es ideal para tener problemas de inyección SQL.

```
// Lo que NO DEBEMOS HACER.¡ No lleva marcadores! Inyección SQL!
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) values ($nombre, $clave, $rol)");
```

Insertar datos

Para insertar simplemente hay que ejecutar la sentencia SQL, correspondiente.

Ejemplo, **sin sentencia preparada:**

```
$ins = "insert into usuarios(nombre, clave, rol) values('Susana', '44444', '2');";
$resul = $bd->query($ins);

    // errores
if($resul) {
    echo "Se ha añadido un nuevo usuario correctamente <br>";
    echo "Filas insertadas: " . $resul->rowCount() . "<br>";
}else print_r( $bd -> errorinfo());
// para los autoincrementos
echo "Código de la fila insertada " . $bd->lastInsertId() . "<br>";
```

Lo aconsejable es usar siempre una sentencia preparada.

Como ya hemos visto, cuando se **obtienen, insertan o actualizan datos con PDO se suele seguir el esquema**

PREPARE -> [BIND] -> EXECUTE.

Siguiendo este esquema y continuando con el ejemplo de partida, vamos a **insertar con sentencias preparadas.**

A. Ejemplo de **inserción** con consultas preparadas y **usando marcadores anónimos o de posición.**

a.1. Preparamos la consulta

```
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol)
                     values (?, ?, ?)");
```

Con esta sentencia sólo hemos preparado la consulta, el siguiente paso será vincular los marcadores anónimos o de posición con los valores que seguramente habremos obtenido a través de un formulario.

a.2. Vinculamos los marcadores.

Para **vincular** los marcadores anónimos con su correspondiente valor se puede utilizar **bindParam** o **bindValue**.

La **diferencia** entre el uso de uno u otro es que:

- Con bindParam() se vincula la variable al parámetro y en el momento de hacer execute() se asigna realmente el valor de la variable a ese parámetro.
- Con bindValue se asigna el valor de la variable a ese parámetro justo en el momento de ejecutar la instrucción bindValue. Es decir, se enlaza el valor de la variable y permanece hasta execute().

En la práctica bindValue() se suele usar cuando se tienen que insertar datos sólo una vez, y bindParam() cuando se tienen que pasar datos múltiples (desde un array por ejemplo).

Ambas funciones aceptan un tercer parámetro, que define el tipo de dato que se espera.

Los tipos de datos más utilizados son:

```
PDO::PARAM_BOOL(booleano)
PDO::PARAM_NULL(null)
```

```
PDO::PARAM_INT(integer)
PDO::PARAM_STR (string)
```

Por ejemplo:

```php
$stmt->bindParam(':edad', $miedad, PDO::PARAM_INT);
$stmt->bindParam(':apellidos', $misApellidos, PDO::PARAM_STR, 40);  // 40 caracteres como máximo.
```

Siguiendo con nuestro ejemplo, lo primero que haremos será asignar a cada uno de los marcadores anónimo una variable, y además lo haremos indexándolos con números:

```php
$ins->bindParam(1, $nombre);
$ins->bindParam(2, $passw);
$ins->bindParam(3, $rol);
```

a.3 Ejecutamos la sentencia preparada tantas veces como deseemos:

```php
// Insertamos una fila.
$nombre = "Daniel";
$passw = "777777";
$rol= "1";
$ins->execute();

//Insertamos otra fila con valores diferentes.
$nombre = "Andrea";
$passw = "888888";
$rol= "0";
$ins->execute();
```

También es posible asignar los marcadores anónimos usando un array asociativo donde pondremos los datos que deseamos insertar. La mayoría de las veces esos datos se han obtenido directamente desde el formulario directamente en un array:

```php
$datos = array('Pedro', '666666', '0');
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) values (?, ?, ?)");
$ins->execute($datos);
```

B. Insertar utilizando **marcadores conocidos** o **por nombre**.

Es la forma más recomendable de trabajar con PDO, ya que a la hora de hacer el bindParam o el bindValue se puede especificar el tipo de datos y la longitud máxima de los mismos.

```
// Preparamos la consulta
$stmt = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) VALUES
(:nombre, :clave, :rol)");


// vinculamos los marcadores


$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':clave', $clave);
$stmt->bindParam(':rol', $rol);


// insertamos un usuario nuevo
$nombre = "Carmen";
$clave = "25252525";
$rol = "1";


$stmt->execute();
```

Al igual que en el ejemplo anterior, podemos asignar los marcadores anónimos a través de un array asociativo. Por ejemplo:

```
//Datos a insertar
$datos = array('nombre' => 'Olga', 'clave' => '676767','rol'=> '0');


//Preparamos la consulta


$stmt  =  $bd->prepare("INSERT  INTO  usuarios(nombre,  clave,  rol)  values
(:nombre,:clave, :rol)");


//ejecutamos


$stmt->execute($datos); //se pueden poner directamente los valores aquí
```

Nunca uses la sentencia **prepare sin marcadores** porque permitiría realizar inyección SQL. Por ejemplo, NO LA USES ASÍ:

```
    $ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) values
                         ($nombre,$clave, $rol)");
```

Los **marcadores conocidos** nos permiten trabajar con **objetos**, directamente en la base de datos, asumiendo que las propiedades de ese objeto coinciden con los nombres de los campos de la tabla en la base de datos.

Ejemplo:

```
class Usuarios
  {
     public $nombre;
     public $clave;
     public $rol;
     public function __construct($nombre, $clave, $rol){
                $this->nombre = $nombre;
                $this->clave = $clave;
                $this->rol = $rol;
           }
     // ...etc. Código de la clase....
   }


 $usu = new Usuarios('Margarita', '3456789','2');
 $stmt = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) VALUES (:nombre, :clave, :rol)");
 if($stmt->execute((array) $usu)){
            echo "Se ha creado un nuevo registro";
 };
```

Modificar datos

Ejemplos:

```
//Ejemplo 1: ACTUALIZAR sin sentencia preparada
echo 'Ejemplo 1:<br>';
$upd = "update usuarios set rol = 1 where rol = 0";
$resul = $bd->query($upd);
//comprobar errores
if($resul){
        echo "update correcto <br>";
        echo "Filas actualizadas: " .$resul->rowCount(). "<br>";
}else print_r( $bd -> errorinfo());
```

```php
// Ejemplo 2 con SENTENCIA PREPARADA
try{
    // Modificar datos de los usuarios con SENTENCIA PREPARADA Marcadores por NOMBRE

    $stmt = $bd->prepare('UPDATE usuarios SET rol = :rol WHERE nombre = :nombre');
    // asignamos los marcadores

    $stmt->bindParam(':nombre', $nombre);
    $stmt->bindParam(':rol', $rol);
    // Indicamos el usuario y los datos a modificar
    $nombre = "Carmen";
    $rol = "4";

    $stmt->execute();
    echo "Filas actualizadas: " .$stmt->rowCount();
}
catch (PDOException $err)
{
    die("Error ejecutando modificación SQL. ".$err->getMessage());
}
```

Borrar datos

Ejemplos:

```php
    // borrar sin sentencias preparadas

    echo 'Ejemplo 1: borramos el usuario Margarita <br>';

    $del = "delete from usuarios where nombre = 'Margarita'";

    $resul = $bd->query($del);

    //comprobar errores

    if($resul){

        echo "Borrado correcto <br>";

        echo "Filas borradas: " . $resul->rowCount() . "<br>";

    }else print_r( $bd -> errorinfo());
```

```php
// Ejemplo 2 borrar con sentencia preparada

  // borramos el usuario Angelines

try{

    // Borrar usuarios con SENTENCIA PREPARADA Marcadores por NOMBRE

    $stmt = $bd->prepare('DELETE FROM usuarios WHERE nombre = :nombre');
    // asignamos los marcadores

    $stmt->bindParam(':nombre', $nombre);

    // Indicamos el usuario a borrar
    $nombre = "Angelines";

    $stmt->execute();
    echo "Filas borradas: " .$stmt->rowCount();
}
catch (PDOException $err)
{
    die("Error ejecutando modificación SQL. ".$err->getMessage());
}
```

Consultas SELECT preparadas

Los datos que resultan de realizar consultas SELECT se obtienen a través del método fetch() o del método fetchAll().

- **->fetch()** : Obtiene la siguiente fila de un conjunto de resultados
- **->fetchAll()**: Devuelve un array que contiene todas las filas del conjunto de resultados (el tipo de datos a devolver se puede indicar como parámetro).

Antes de llamar a fetch (o durante) hay que especificar como se quieren devolver los datos:

- **PDO::FETCH_ASSOC**: devuelve un array indexado por el nombre de las columnas del conjunto de resultados de la tabla.
- **PDO::FETCH_NUM**: Devuelve un array indexado por el número de columna tal como fue devuelto en el conjunto de resultados, comenzando por la columna 0.
- **PDO::FETCH_BOTH**: valor por defecto. Devuelve un array indexado tanto por nombre de columna, como numéricamente con índice de base 0 tal como fue devuelto en el conjunto de resultados.

- **PDO::FETCH_BOUND**: Devuelve TRUE y asigna los valores de las columnas del conjunto de resultados a las variables de PHP a las que fueron vinculadas con el con el método PDOStatement::bindColumn.
- **PDO::FETCH_CLASS**: Devuelve una nueva instancia de la clase solicitada, haciendo corresponder las columnas del conjunto de resultados con los nombres de las propiedades de la clase. Creará las propiedades si éstas no existen.
- **PDO::FETCH_INTO**: Actualiza una instancia existente de la clase solicitada, haciendo coincidir el nombre de las columnas con los nombres de las propiedades de la clase..
- **PDO::FETCH_OBJ**: devuelve un objeto anónimo con nombres de propiedades que corresponden a las columnas.
- **PDO::FETCH_LAZY**: combina **PDO::FETCH_BOTH** y **PDO::FETCH_OBJ**, creando los nombres de las propiedades del objeto tal como se accedieron.

Ejemplos de consultas preparadas.

A. Consulta **SELECT preparada sin parámetros**:

```php
<?php
//Ejemplo de consulta preparada y sin parámetros
$servername = "localhost";
$database = "empresa";
$username = 'root';
$password = '';
try {
    $bd = new PDO("mysql:host=$servername;dbname=$database;charset=utf8", $username, $password);
    $bd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    try
    {
        $stm = $bd->prepare("SELECT * FROM usuarios");
        $stm->execute();
        foreach ($stm->fetchAll(PDO::FETCH_ASSOC) as $row){
            echo "Nombre: " . $row["nombre"] . ", Rol: " .$row["rol"] . "<br>";
        }
    }
    catch (PDOException $err)
    {
        die("Error ejecutando consulta SQL. ".$err->getMessage());
    }


} catch (PDOException $e) {
    die('Error con la base de datos: ' .$e->getMessage());

}

// Liberamos los recursos utilizados
$stmt=null;

// Se recomienda cerrar la conexión para liberar recursos de forma más rápida.
$bd = null;

?>
```

B. **SELECT preparada**, con **marcadores anónimos** o de **posición**.

```php
// $preparada sería un objeto de tipo PDOStatement
try
    {
        $preparada = $bd->prepare("select nombre from usuarios where rol = ?");
        $preparada->bindParam(1, $rol);
        $rol= 2;
        $preparada->execute();

        //Leemos los datos del recordset que nos devuelve SELECT en el objeto PDOStatement.
        $preparada->setFetchMode(PDO::FETCH_ASSOC);
        while($row = $preparada->fetch()){
            echo "<p> Nombre: " . $row['nombre'] . "<p/>";
        }
        echo "<p><strong>Número de usuarios con rol ". $rol." : </strong>" .  $preparada->rowCount()."</p>";


    }
catch (PDOException $err)
    {
        die("Error ejecutando consulta SQL. ".$err->getMessage());
    }
```

c. **SELECT preparada** con **marcadores conocidos** o **POR NOMBRE**. En este caso se usarán esos nombres como clave del array de execute().

```php
try
    {
        $preparada= $bd->prepare("select nombre from usuarios where rol = :rol");
        $preparada->bindParam(':rol', $rol);
        $rol= 2;
        $preparada->execute();
        //Leemos los datos del recordset que nos devuelve SELECT en el objeto PDOStatement.
        $data = $preparada->fetchAll(PDO::FETCH_ASSOC);//se obtienen todas las filas
        foreach($data as $row){
        echo "Nombre: ".$row['nombre']."<br>";
        }
        echo "<p><strong>Número de usuarios con rol ". $rol." : </strong>" .  $preparada->rowCount()."</p>";

    }
catch (PDOException $err)
    {
        die("Error ejecutando consulta SQL. ".$err->getMessage());
    }
```

Se propone repetir estos ejemplos usando PDO::FETCH_OBJ

Otros métodos interesantes:

- **lastInsertId():** Devuelve el id del último registro insertado en la tabla. Es un método de PDO `$ido->lastInsertId();`
- **quote():** Si no usas consultas preparadas, una forma de protegerte contra la inyección SQL es usando este método:

  `$sqlSegura= $pdo->quote($sqlInsegura);`
- **rowCount():**Devuelve un entero indicando el número de filas afectadas por la última operación.

  `$rows_affected = $stmt->rowCount();`

Para evitar tener repetida la configuración del acceso a la base de datos se sugiere crear un archivo, donde declaremos los parámetros de la conexión como constantes, así las tendremos disponibles en toda la aplicación. Incluso podemos crear la clase BaseDatos con dicho código, añadiendo un método para realizar consultas y otro para extraer registros de la base de datos.

Ejemplo:

```php
<?php
namespace Lib;
use PDO;
class BaseDatos extends PDO{
    private PDO $conexion;
    private mixed $resultado;
    public function __construct(
            private $tipo_de_base = 'mysql',
            private string $servidor = SERVIDOR,
            private string $usuario = USUARIO,
            private string $pass = PASS,
            private string $base_datos= BASE_DATOS) {
      //Sobreescribo el método constructor de la clase PDO.
      try{
          $opciones = array(
            PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
            PDO::MYSQL_ATTR_FOUND_ROWS => true
            );
         parent::__construct("{$this->tipo_de_base}:dbname={$this->base_datos};host={$this->servidor}", $this->usuario, $this->pass, $opciones);
      }catch(PDOException $e){
         echo 'Ha surgido un error y no se puede conectar a la base de datos. Detalle: ' . $e->getMessage();
         exit;
      }
   }


}
```

Otro ejemplo:

```php
<?php
namespace Lib;
use PDO;

class BaseDatos {

  private PDO $conexion;
  private mixed $resultado; //mixed novedad en PHP cualquier valor

  function __construct(
    private string $servidor = SERVIDOR,
    private string $usuario = USUARIO,
    private string $pass = PASS,
    private string $base_datos= BASE_DATOS
  ) {
    $this->conexion = $this->conectar();
  }

  private function conectar(): PDO {
    try {
        $opciones = array(
            PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
            PDO::MYSQL_ATTR_FOUND_ROWS => true
            );
        $conexion = new PDO("mysql:host={$this->servidor};dbname={$this->base_datos}", $this->usuario, $this->pass, $opciones);
        return $conexion;
    } catch(PDOException $e){
      echo 'Ha surgido un error y no se puede conectar a la base de datos. Detalle: ' . $e->getMessage();
      exit;
    }
  }

   public function consulta(string $consultaSQL): void {
       $this->resultado = $this->conexion->query($consultaSQL);
  }
  public function extraer_registro(): mixed {
    return ( $fila = $this->resultado->fetch(PDO::FETCH_ASSOC ))? $fila:false;
```