UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL AVANZADO

www.sceu.frba.utn.edu.ar/e-learning

Módulo II

CONTINUAMOS CON PHP AVANZADO

Sentencias Preparadas Mysqli y PDO

Presentación de la Unidad:

En esta unidad veremos como trabajar con sentencias preparadas de Mysql.

Las dos extensiones para trabajar con Mysql, Mysqli y PDO de PHP ofrecen la posibilidad de trabajar con sentencias preparadas que ofrece entre otras ventajas una mayor seguridad en nuestro código.

Veremos como funcionan estas sentencias.

Objetivos:

- Aprender a utilizar sentencias preparadas con la extensión Mysqli.
- Aprender a utilizar las sentencias preparadas de la extensión PDO.

Temario:

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

SENTENCIAS PREPARADAS

Una **Sentencia Preparada**, o **Prepared Statement**, es una característica de algunos **sistemas de bases de datos** que permite ejecutar la misma (o similar) **sentencia SQL** provocando una mejora en la seguridad y en el rendimiento de la aplicación.

Las bases de datos MySQL soportan sentencias preparadas. Una sentencia preparada o una sentencia parametrizada se usa para ejecutar la misma sentencia repetidamente con gran eficiencia.

Flujo de trabajo básico

La ejecución de sentencias preparadas consiste en dos etapas: la preparación y la ejecución. En la etapa de preparación se envía una plantilla de sentencia al servidor de bases de datos. El servidor realiza una comprobación de sintaxis e inicializa los recursos internos del servidor para su uso posterior.

El servidor de MySQL soporta el uso de parámetros de sustitución posicionales anónimos con *?*.

1. Cómo funcionan las sentencias preparadas

- **Prepare**. Primero una **plantilla de la sentencia SQL** se crea y se envía a la base de datos. Algunos valores se dejan sin especificar, llamados **parámetros** y representados por un **interrogante** "**?**":

```
INSERT INTO Clientes VALUES (?, ?, ?, ?);
```

- Después, la base de datos analiza, compila y realiza la optimización de la consulta sobre la sentencia SQL, y **guarda el resultado sin ejecutarlo**.
- **Execute**. Por último, **la aplicación enlaza valores con los parámetros**, y la base de datos ejecuta la sentencia. La aplicación puede entonces ejecutar la sentencia tantas veces como quiera con valores diferentes.

2. Ventajas de las Sentencias Preparadas

- Las **Sentencias Preparadas** reducen en **tiempo de análisis** ya que la preparación de la consulta de realiza sólo una vez (aunque la sentencia se ejecute las veces necesarias).
- Los **parámetros** enlazados minimizan el ancho de banda consumida del servidor ya que sólo necesitas enviar los parámetros cada vez, no la consulta entera.

- Las **Sentencias Preparadas** son muy últiles frente a Inyecciones SQL, ya que los valores de los parámetros, que son transmitidos después usando un protocolo diferente, no necesitan ser escapados. Si la plantilla de la sentencia original no es derivada de un **input externo**, los **ataques SQL Injection** no pueden ocurrir.

3. Sentencias Preparadas en MySQLi

Vamos a ver un **ejemplo de Sentencias Preparadas en PHP con MySQLi**.

```
$server = "localhost";
$user = "usuario";
$password = "password";
$dbname = "ejemplo";
// Conectar
$db = new mysqli($server, $user, $password, $dbname);
// Comprobar conexión
if($db->connect_error){
    die("La conexión ha fallado, error número " . $db->connect_errno .
": " . $db->connect_error);
}
```

Ya tenemos la conexión a la base de datos, ahora podemos utilizarla en los archivos donde tengamos que hacer sentencias.

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (?, ?, ?)");
$stmt->bind_param('ssi', $nombre, $ciudad, $contacto);
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute();
$nombre = "Hillary Clinton";
$ciudad = "Barcelona";
$contacto = 4665767;
$stmt->execute();
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$stmt->close();
$db->close();
```

Incluimos un interrogante donde queremos sustituir un ***integer***, un ***string***, un ***double*** o un ***blob*** en la función *prepare()*. Después usamos la función _bind_param()_:

```
$stmt->bind_param('ssi', $nombre, $ciudad, $contacto);
```

La función enlaza los **parámentros** con la **consulta SQL** y le dice a la **base de datos** que parámetros son. El argumento "ssi" especifica el **tipo de dato** que se espera que sea el parámetro. Pueden ser de cuatro tipos:

Letra	Tipo de dato
i	Integer

d double

s string

b blob

Se debe especificar uno por cada parámetro.

La función _mysqli_stmt::bind_result()_ permite enlazar las columnas con variables para su uso posterior. No hace falta especificar el tipo de dato.

```
// Iniciamos sentencia preparada
if ($stmt = $mysqli->prepare("SELECT nombre, ciudad FROM Clientes")) {
    $stmt->execute();
    // Vinculamos variables a columnas
    $stmt->bind_result($nombre, $ciudad);
    // Obtenemos los valores
    while ($stmt->fetch()) {
        printf("%s %s\n", $nombre, $ciudad);
    }
    // Cerramos la sentencia preparada
    $stmt->close();
}
```

Tienen mas información sobre sentencias preparadas y especificaciones técnicas en el manual de PHP: http://php.net/manual/es/mysqli.quickstart.prepared-statements.php

4. Sentencias Preparadas en PDO

Vamos a hacer el mismo ejemplo anterior pero con **PDO**.

```
$server = "localhost";
$user = "usuario";
$password = "password";
$dbname = "ejemplo";
try {
// Conectar
$db = new PDO("mysql:host=$server;dbname=$dbname", $user, $password);
// Establecer el nivel de errores a EXCEPTION
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e){
    echo "Error: " . $e->getMessage();
}
```

Una vez establecida la conexión, podemos usarla donde la necesitemos:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (:nombre, :ciudad, :contacto)");
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':ciudad', $ciudad);
$stmt->bindParam(':contacto', $contacto);
// Establecer parámetros y ejecutar
```

```
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute();
$nombre = "Hillary Clinton";
$ciudad = "Barcelona";
$contacto = 4665767;
$stmt->execute();
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```

Los parámetros también pueden enlazarse sin utilizar *bindParam()*, directamente con un array en *execute()*:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (:nombre, :ciudad, :contacto)");
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute(array(':nombre' => $nombre, ':ciudad' => $ciudad,
':contacto' => $contacto));
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```

Hemos asignado los parámetros con nombre de variable, pero también se puede hacer con números:

```
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (?, ?, ?)");
$stmt->bindParam(1, $nombre);
$stmt->bindParam(2, $ciudad);
$stmt->bindParam(3, $contacto);
```

En caso de pasar los parámetros a través de un array con *execute()*, se haría de la misma forma que antes:

```
// Preparar
$stmt = $db->prepare("INSERT INTO Clientes (nombre, ciudad, contacto)
VALUES (?, ?, ?)");
// Establecer parámetros y ejecutar
$nombre = "Donald Trump";
$ciudad = "Madrid";
$contacto = 4124124;
$stmt->execute(array($nombre, $ciudad, $contacto));
// Mensaje de éxito en la inserción
echo "Se han creado las entradas exitosamente";
// Cerrar conexiones
$db = null;
```

EJEMPLO PREPARE MYSQLI PARA SELECT E INSERT

En el archivo prepare.php encontraran dos ejemplos de la utilización de las sentencias preparadas con un select y un insert.

La tarea propuesta es que hagan el update y el delete con sentencias preparadas. En el caso del update los interrogantes estarán en los valores a modificar y en la condicion. Por ejemplo

$stmt = $mysqli->prepare("UPDATE usuarios SET nombre = ? WHERE id = ? ")

En el caso del DELETE los interrogantes estarán en la condición WHERE.

EJEMPLO PRACTICO CRUD PHP, PDO Y AJAX

Vamos a ver un ejemplo completo de aplicación CRUD utilizando las sentencias preparadas de PDO.

Este ejemplo esta desarrollado con los métodos de Jquery para el uso de Aax y Bootstrap para el diseño de las vistas. Mas alla de las características de JQuery que se utilizan que no nos detendremos mucho en explicar porque no es el objetivo de este curso presten atención a los scripts PHP que trabajan contra la base de datos para dar de alta, baja, modificar o consultar los datos.

En el archivo db_connection.php encontraran la conexión a la base de datos utilizando PDO.

En el archivo Ajax/lib.php encontraran los métodos que realizan las acciones contra la base de datos para dar de alta, baja, modificar y consultar.

```
<?php

require __DIR__ . '/db_connection.php';

class CRUD
{

   protected $db;

   function __construct()
   {
      $this->db = DB();
   }

   function __destruct()
   {
      $this->db = null;
   }
```

```
    /*
     * Add new Record
     *
     * @param $first_name
     * @param $last_name
     * @param $email
     * @return $mixed
     * */
    public function Create($first_name, $last_name, $email)
    {
        $query = $this->db->prepare("INSERT INTO users(first_name, last_name, email)
VALUES (:first_name,:last_name,:email)");
        $query->bindParam("first_name", $first_name, PDO::PARAM_STR);
        $query->bindParam("last_name", $last_name, PDO::PARAM_STR);
        $query->bindParam("email", $email, PDO::PARAM_STR);
        $query->execute();
        return $this->db->lastInsertId();
    }

    /*
     * Read all records
     *
     * @return $mixed
     * */
    public function Read()
    {
        $query = $this->db->prepare("SELECT * FROM users");
        $query->execute();
        $data = array();
        while ($row = $query->fetch(PDO::FETCH_ASSOC)) {
            $data[] = $row;
        }
        return $data;
    }

    /*
     * Delete Record
     *
     * @param $user_id
     * */
    public function Delete($user_id)
    {
        $query = $this->db->prepare("DELETE FROM users WHERE id = :id");
        $query->bindParam("id", $user_id, PDO::PARAM_STR);
        $query->execute();
    }

    /*
     * Update Record
```

```
     *
     * @param $first_name
     * @param $last_name
     * @param $email
     * @return $mixed
     * */
    public function Update($first_name, $last_name, $email, $user_id)
    {
        $query = $this->db->prepare("UPDATE users SET first_name = :first_name,
last_name = :last_name, email = :email  WHERE id = :id");
        $query->bindParam("first_name", $first_name, PDO::PARAM_STR);
        $query->bindParam("last_name", $last_name, PDO::PARAM_STR);
        $query->bindParam("email", $email, PDO::PARAM_STR);
        $query->bindParam("id", $user_id, PDO::PARAM_STR);
        $query->execute();
    }

    /*
     * Get Details
     *
     * @param $user_id
     * */
    public function Details($user_id)
    {
        $query = $this->db->prepare("SELECT * FROM users WHERE id = :id");
        $query->bindParam("id", $user_id, PDO::PARAM_STR);
        $query->execute();
        return json_encode($query->fetch(PDO::FETCH_ASSOC));
    }
}

?>
```

Estos métodos son llamados cada uno desde su correspondiente archivo:

Create.php: llama al método create(…) de la clase CRUD para dar de alta un registro

Read.php: llama al método read() de la clase CRUD que lee los registros de la tabla.

Update.php: llama al método update(…..) de la clase CRUD modifica un registro en la tabla

Delete.php: llama al método delete(…) de la clase CRUD que da de baja el registroen cuestión

Details.php: llama al método details(…) de la clase CRUD para mostrar los detalles de un registro en particular y lo utiliza para mostrarlo al usuario para que modifique los datos.