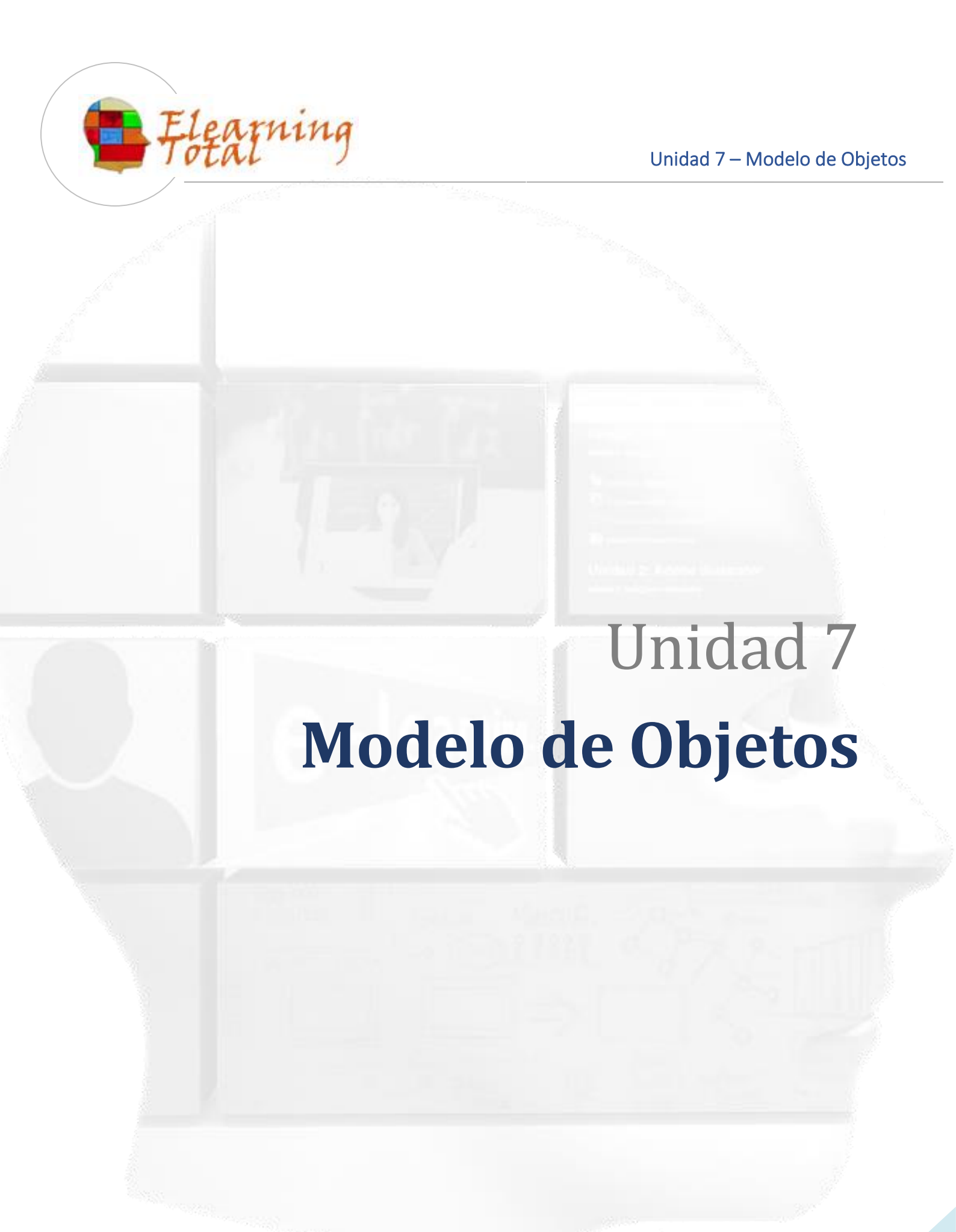

Unidad 7

Modelo de Objetos

Contacto: consultas@elearning-total.com

Web: www.elearning-total.com

Mysqli orientado a objetos.

Ejemplo de ejecución de sentencias

Ejemplo #1 Conectando a MySQL

```
<?php
$mysqli = new mysqli("ejemplo.com", "usuario", "contraseña", "basedato
s");
if ($mysqli->connect_errno) {
 echo "Falló la conexión con MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
 !$mysqli->query("CREATE TABLE test(id INT)") ||
 !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
 echo "Falló la creación de la tabla: (" . $mysqli-
>errno . ") " . $mysqli->error;
}
?>
```

Ejemplo #2 Navegación a través de resultados

```
<?php
$mysqli = new mysqli("ejemplo.com", "usuario", "contraseña", "basedato
s");
if ($mysqli->connect_errno) {
 echo "Falló la conexión a MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
 !$mysqli->query("CREATE TABLE test(id INT)") ||
 !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
 echo "Falló la creación de la tabla: (" . $mysqli-
>errno . ") " . $mysqli->error;
}
$resultado = $mysqli->query("SELECT id FROM test ORDER BY id ASC");
echo "Orden inverso...\n";
for ($num_fila = $resultado->num_rows - 1; $num_fila >= 0; $num_fila--
) {
 $resultado->data_seek($num_fila);
```

```
 $fila = $resultado->fetch_assoc();
 echo " id = " . $fila['id'] . "\n";
}
echo "Orden del conjunto de resultados...\n";
$resultado->data_seek(0);
while ($fila = $resultado->fetch_assoc()) {
 echo " id = " . $fila['id'] . "\n";
}
?>
```

El resultado del ejemplo sería:

```
Orden inverso...
 id = 3
id = 2
id = 1
Orden del conjunto de resultados...
id = 1
id = 2
id = 3
```

Sentencias preparadas

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 ?.

La preparación es seguida de la ejecución. Durante la ejecución el cliente vincula los valores de los parámetros y los envía al servidor. El servidor crea una sentencia desde la plantilla de la sentencia y los valores vinculados para ejecutarla usando los recursos internos previamente creados.

Ejecución repetida:

Una sentencia preparada se puede ejecutar repetidamente. En cada ejecución el valor actual de la variable vinculada se evalúa y se envía al servidor. La sentencia no se analiza de nuevo. La plantilla de la sentencia no es transferida otra vez al servidor.

Cada sentencia preparada ocupa recursos del servidor. Las sentencias deberían cerrarse explícitamente inmediatamente después de su uso. Si no se realiza explícitamente, la sentencia será cerrada cuando el gestor de la sentencia sea liberado por PHP.

Usar una sentencia preparada no es siempre la manera más eficiente de ejecutar una sentencia. Una sentencia preparada ejecutada una sola vez causa más viajes de ida y vuelta desde el cliente al servidor que una sentencia no preparada. Es por esta razón que si debo preparar una sentencia que luego se ejecutará una sola vez conviene ejecutarla directamente a través de mysqli_query().

Veamos un ejemplo:

```
<?php
$mysqli = new mysqli("localhost", "root", "", "login");
/* check connection */
if (mysqli_connect_errno()) {
 printf("Connect failed: %s\n", mysqli_connect_error());
 exit();
}
$user = "juan";
$pass = "juan1234";
/* create a prepared statement
Creo una sentencia preparada en la cual reemplazaré los marcadores ?
por los valores correspondientes con bind->param
*/
if ($stmt = $mysqli->prepare("SELECT nombre, apellido, email, dni FROM
usuarios WHERE usuario=? AND password=?")) {
 /* bind parameters for markers
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [,
mixed &$... ] )
El método bind_param recibe como primer parámetro un string con
los tipos de datos que se enviarán para cada uno de los marcadores
establecidos en el prepare segun las siguientes posibilidades:
i la variable correspondiente es de tipo entero
d la variable correspondiente es de tipo double
s la variable correspondiente es de tipo string
```

```
b la variable correspondiente es un blob y se envía en
paquetes
Y a continuación los valores para los marcadores en el orden en
que deben ser reemplazados
*/
 $stmt->bind_param("ss", $user, $pass);
 /* execute query */
 $stmt->execute();
 /* bind result variables
Vincula variables a la sentencia preparada, en este caso nombre,
apellido, email, dni
*/
 $stmt->bind_result($nombre, $apellido, $email, $dni);
 /* fetch value */
 $stmt->fetch();
 printf("Los datos para el user y pass ingresados son: %s %s %s
%s", $nombre, $apellido, $email, $dni);
 /* close statement */
 $stmt->close();
}
/* close connection */
$mysqli->close(); ?>
```

PDO

La extensión Objetos de Datos de PHP (PDO por sus siglás en inglés) define una interfaz ligera para poder acceder a bases de datos en PHP. Cada controlador de bases de datos que implemente la interfaz PDO puede exponer características específicas de la base de datos, como las funciones habituales de la extensión.

PDO viene con PHP 5.1, y está disponible como una extensión PECL para PHP 5.0; PDO requiere las características nuevas de OO del núcleo de PHP 5, por lo que no se ejecutará con versiones anteriores de PHP.

Conexiones y su administración

Las conexiones se establecen creando instancias de la clase base PDO. No importa el controlador que se utilice; siempre se usará el nombre de la clase PDO. El constructor acepta parámetros para especificar el origen de datos (conocido como DSN) y, opcionalmente, el nombre de usuario y la contraseña (si la hubiera).

Ejemplo #1 Conectarse a MySQL

```
<?php
$gbd = new PDO('mysql:host=localhost;dbname=test', $usuario, $contrase
ña);
?>
```

Si hubiera errores de conexión, se lanzará un objeto PDOException. Se puede capturar la excepción si fuera necesario manejar la condición del error, o se podría optar por dejarla en manos de un gestor de excepciones global de una aplicación configurado mediante set_exception_handler()

Ejemplo #2 Manejo de errores de conexión

```
<?php
try {
 $gbd = new PDO('mysql:host=localhost;dbname=test', $usuario, $cont
raseña);
 foreach($gbd->query('SELECT * from FOO') as $fila) {
 print_r($fila);
 }
 $gbd = null;
} catch (PDOException $e) {
```

```
print "¡Error!: " . $e->getMessage() . "<br/>";
 die();
}
?>
```

Una vez realizada con éxito una conexión a la base de datos, será devuelta una instancia de la clase PDO al script. La conexión permanecerá activa durante el tiempo de vida del objeto PDO. Para cerrar la conexión, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas (esto se puede hacer asignando NULL a la variable que contiene el objeto). Si no se realiza explícitamente, PHP cerrará automáticamente la conexión cuando el script finalice.

Ejemplo #3 Cerrar una conexión

```
<?php
$gbd = new PDO('mysql:host=localhost;dbname=test', $usuario, $contrase
ña);
// Use la conexión aquí
// ya se ha terminado; la cerramos
$gbd = null;
?>
```

Veamos un ejemplo completo de conexión y consulta de los datos de una tabla utilizando los métodos de la clase:

```
<?php
$dsn = 'mysql:dbname=personas;host=127.0.0.1';
$usuario = 'root';
$contraseña = '';
try {
 $gbd = new PDO($dsn, $usuario, $contraseña);
} catch (PDOException $e) {
 echo 'Falló la conexión: ' . $e->getMessage();
}
$result = $gbd->query("select * from personas");
$row = $result->fetchAll();
var_dump($row);
foreach ($row as $fila){
echo $fila['id_persona'].'<br />';
}
var_dump($row);
?>
```