



# PHP & MySQL

*Anexo 8.- PDO*



DISTRIBUIDO POR:

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

## PDO

PHP posee multitud de extensiones para acceder específicamente a diferentes gestores de bases de datos tales como SQL Server, MySQL, PostgreSQL, IBM DB2..., etc.

La documentación de estas extensiones puede encontrarse en la web oficial de PHP:

<http://php.net/manual/es/refs.database.vendors.php>

A diferencia de estas extensiones específicas, PDO ofrece un modo único de acceder a cualquier sistema de base de datos empleando la misma extensión de PHP. Esto constituye una *capa de abstracción de acceso a datos*, que permite que el código de la aplicación web sea independiente del tipo de base de datos y pueda reutilizarse para cualquier otro tipo (MySQL, SQL Server, SQLite..., etc ).

### Establecimiento de conexión

El establecimiento de conexión consiste en crear un objeto de la clase PDO indicando tres argumentos en el constructor:

```
public PDO::__construct ( string $dsn  
                        [, string $username  
                        [, string $password  
                        [, array $options ]]] )
```

El primer parámetro *\$dsn* es una cadena que determinan la conexión al servidor de base de datos y la base de datos a la que se quiere conectar:

```
$host = '192.168.9.180'; // IP del servidor  
$db   = 'banco';        // Base de datos  
$port = 3306;           // Puerto  
$charset = 'utf8';      // Set de caracteres  
  
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
```

**(\*) Si el servidor de base de datos está instalado en nuestra propia máquina, puede indicarse como IP el valor 127.0.0.1.**

**Ejemplo:** Valor de cadena *\$dsn* construida:

```
mysql:host=192.168.9.180;dbname=banco;charset=utf8;port=3306
```

Esta DSN determina la conexión a una base de datos *banco* almacenada en un servidor de MySQL situado en la IP: 192.168.9.180. El parámetro **charset** determina que se va a emplear la codificación 'UTF8' para el conjunto de caracteres ( compatible con signos como 'ñ', tildes..., etc ). El puerto 3306 es el predeterminado para los servidores de MySQL y puede omitirse.

La DSN no debe tener espacios en blanco, ni ningún otro carácter de adorno.

Los parámetros *\$username* y *\$password* indican el nombre de usuario y la contraseña de la cuenta con permisos de acceso al servidor y la base de datos. El último parámetro *\$options* es una matriz asociativa de opciones de configuración.

**Ejemplo:** Creación de un objeto PDO:

```
$host = '192.168.9.180'; // IP del servidor
$db = 'banco'; // Base de datos
$port = 3306; // Puerto
$charset = 'utf8'; // Set de caracteres
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";

$user = 'alumno'; // Usuario
$pass = 'cipsa'; // Contraseña

// Opciones de configuracion
$opt = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false,
];

// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
```

El objeto PDO representa una conexión con el servidor de base de datos. Dado que el nº de conexiones que soporta un SGBD es limitado; TODAS las operaciones contra la base de datos deben realizarse a partir de un mismo objeto, en vez de crear un nuevo objeto para cada operación.

### Ejecución de comandos.

Las sentencias SQL para la base de datos pueden realizarse empleando dos métodos de la clase PDO:

**Método *query()*.**

```
public PDOStatement PDO::query ( string $statement )
```

El método ***query()*** ejecuta la sentencia SQL dada como parámetro (*\$statement*) y devuelve un objeto **PDOStatement** que representa *el conjunto de resultados* generado por la consulta. En caso de producirse un error, retorna un valor *FALSE*.

```
<?php
// Obtencion conjunto de resultados
$resultados = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM CUENTAS");
?>
```

El conjunto de resultados puede recorrerse para obtener uno a uno los registros devueltos por la consulta invocando al método ***fetch()***. Este método devuelve una matriz asociativa con los nombres de las columnas y los valores de los campos de un registro. En caso no haber más registros en el conjunto de resultados retorna *FALSE*.

**Ejemplo:** El siguiente código obtiene y muestra todos los campos de todos los registros almacenados en la tabla CUENTAS:

```
<?php

$host = '192.168.9.180';           // IP del servidor
$db   = 'banco';                  // Base de datos
$port = 3306;                    // Puerto
$charset = 'utf8';               // Set de caracteres
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";

$user = 'alumno';                // Usuario
$pass = 'cipsa';                // Contraseña

// Opciones de configuracion
$opt = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);

// Ejecucion consulta -> obtención conjunto de resultados
$resultados = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM
CUENTAS");

// Bucle de recorrido de conjunto de resultados
while ($reg = $resultados->fetch())
{
    var_dump($reg); // Visualizacion matriz de valores de registro
}

?>
```

```
C:\xampp7\htdocs\prueba\database\base.php:27:
array (size=4)
  'NUMERO_CUENTA' => string 'AC129384' (length=8)
  'TITULAR'      => string 'ROGER PETROV' (length=12)
  'SALDO'        => string '1200.00' (length=7)
  'INTERES'      => string '5.00' (length=4)
C:\xampp7\htdocs\prueba\database\base.php:27:
array (size=4)
  'NUMERO_CUENTA' => string 'AC439586' (length=8)
  'TITULAR'      => string 'YURI PISKUNOV' (length=13)
  'SALDO'        => string '550.00' (length=6)
  'INTERES'      => string '0.00' (length=4)
```

### Ejecución de sentencias parametrizadas

La ejecución de sentencias directas sólo es adecuada cuando la sentencia no depende de otras variables ni de ningún valor externo. Si deseamos por ejemplo obtener los datos de una determinada cuenta bancaria dado por el nº de cuenta determinado, la consulta sería:

```
$sql = "SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
FROM CUENTAS
WHERE NUMERO_CUENTA = '$cuenta'";
```

Una *consulta parametrizada* define marcadores de posición donde deben indicarse valores en vez de concatenar las variables.

**Ejemplo 1:** Consulta parametrizada con marcador de posición sin nombre:

```
$sql = "SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
      FROM CUENTAS
      WHERE NUMERO_CUENTA = ?");
```

**Ejemplo 2:** Consulta parametrizada con marcador de posición con nombre:

```
$sql = "SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
      FROM CUENTAS
      WHERE NUMERO_CUENTA = :cuenta");
```

Al ejecutar la consulta parametrizada los marcadores de posición presentes en el comando se sustituyen por valores. Esto permite reutilizar el comando indicando valores diferentes para los marcadores de posición.

La ejecución de una consulta parametrizada se realiza en dos pasos:

1.- Se declara la sentencia mediante el método **prepare()** del objeto PDO.

```
public PDOStatement PDO::prepare ( string $statement
                                   [, array $driver_options = array() ] )
```

El método recibe como parámetro obligatorio una cadena con el comando SQL parametrizado, y retorna un objeto **PDOStatement**. Si el comando SQL no es correcto sintácticamente retorna un valor lógico FALSO.

2.- Se ejecuta la sentencia llamando al método **execute()** del objeto **PDOStatement** devuelto por el método **prepare()** anterior.

```
public bool PDOStatement::execute ([ array $input_parameters ] )
```

El método recibe como parámetro una matriz con los valores correspondientes a los marcadores de posición del comando SQL parametrizado indicado en **prepare()**. Retorna un valor lógico indicando si la ejecución del comando fue correcta.

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
// Declaración sentencia parametrizada empleando
// marcadores de posición sin nombre.
$resultados = $pdo->prepare("SELECT NUMERO_CUETA, TITULAR, SALDO, INTERES FROM
CUENTAS WHERE NUMERO_CUENTA = ?");
// Ejecución indicando valores para parámetros
$resultados->execute(['AC129384']);
```

En el caso de emplear marcadores de posición sin nombre, la matriz de valores para el método **prepare()** es escalar indicando los valores en el mismo orden en que aparecen los marcadores en el comando SQL. Es decir; primer valor para primera '?', segundo valor para la segunda '?', y así sucesivamente.

## Programación Web en PHP

En caso de emplear marcadores de posición con nombre, la matriz de valores es asociativa indicando como clave el nombre del marcador de posición y su valor:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);

// Declaración sentencia parametrizada empleando
// marcadores de posición sin nombre.
$resultados = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM
CUENTAS WHERE NUMERO_CUENTA = :cuenta");

// Ejecución indicando valores para parámetros
$resultados->execute([':cuenta'=>'AC129384']);
```

**Ejemplo:** El siguiente código muestra un formulario en el que el usuario debe indicar el nº de cuenta. Al pulsar en el botón “**OBTENER**” se muestra los datos de la cuenta:

```
<?php
if ( isset($_POST['ncuenta'])) {
    // Obtencion del valor indicado.
    $cuenta = $_POST['ncuenta'];
    $host = '192.168.9.180';           // IP del servidor
    $db = 'banco';                     // Base de datos
    $port = 3306;                      // Puerto
    $charset = 'utf8';                 // Set de caracteres
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
    $user = 'alumno';                 // Usuario
    $pass = 'cipsa';                  // Contraseña
    // Opciones de configuracion
    $opt = [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false,
    ];
    // Instancia de PDO.
    $pdo = new PDO($dsn, $user, $pass, $opt);
    // Declaración sentencia parametrizada empleando
    $resultados = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM
CUENTAS WHERE NUMERO_CUENTA = :cuenta");
    // Ejecución indicando valores para parámetros
    $resultados->execute([':cuenta'=>$cuenta]);
}

?>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <form method="post">
        N.Cuenta: <input type="text" name="ncuenta">
        <input type="submit" value="Obtener">
    </form>

    <table style='width: 100%'>
    <tr><td>NCUENTE</td><td>TITULAR</td><td>SALDO</td><td>INTERES</td>

        <?php if ( isset($resultados) ) {
            while ( $reg = $resultados->fetch() ) {
                ?>
                <tr><td><?= $reg['NUMERO_CUENTA'] ?></td>
                <td><?= $reg['TITULAR'] ?></td>
                <td><?= $reg['SALDO'] ?></td>
                <td><?= $reg['INTERES'] ?></td></tr>
                <?php
            }
        }

    </table>

</body>
</html>
```

## Limitaciones de la parametrización

Los marcadores de posición no pueden emplearse para sustituir ni palabras clave de los comandos ( **SELECT**, **FROM**, **WHERE**.. ), ni identificadores de columnas o tablas; solo valores de tipo cadena / numéricos.

### Parametrización con partícula LIKE

La partícula LIKE suele emplearse para realizar búsqueda por el valor parcial de un campo de tipo cadena empleando caracteres comodín ( % ).

**Ejemplo:** Supóngase que deseamos obtener todas las cuentas bancarias cuyo titular se apellide “RAMIREZ”. La consulta podría expresarse del siguiente modo:

```
SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM CUENTAS WHERE TITULAR LIKE '%RAMIREZ%'
```

La forma correcta de parametrizar esta consulta sería sustituir todo el valor de la sentencia **LIKE** por un marcador de posición.

(\*) En la parte HTML debe sustituirse el name de la caja de texto por el valor 'titular'.

```
<?php
if ( isset($_POST['titular'])) {
    // Obtencion del valor indicado.
    $titular = $_POST['titular'];
    $host = '192.168.9.180';           // IP del servidor
    $db = 'banco';                    // Base de datos
    $port = 3306;                     // Puerto
    $charset = 'utf8';                // Set de caracteres
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
    $user = 'alumno';                 // Usuario
    $pass = 'cipsa';                  // Contraseña
    // Opciones de configuracion
    $opt = [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false,
    ];
    // Instancia de PDO.
    $pdo = new PDO($dsn, $user, $pass, $opt);
    // Declaración sentencia parametrizada empleando
    // marcadores de posicion sin nombre.
    $resultados = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
FROM CUENTAS WHERE TITULAR LIKE :titular");
    // Ejecución indicando valores para parámetros
    $resultados->execute([':titular'=>"%$titular%"]);
}
?>
```



## Vinculación de parámetros de entrada

Se denomina vinculación al proceso por el cual se vinculan el valor de una determinada variable a un marcadores de posición de una consulta parametrizada. Para ello se emplea el método ***bindParam()*** del objeto ***PDOStatement***.

```
public bool PDOStatement::bindParam ( mixed $parameter ,  
    mixed &$variable  
    [, int $data_type = PDO::PARAM_STR  
    [, int $length  
    [, mixed $driver_options ]]] )
```

Los parámetros del método son los siguientes:

- ***\$parameter*** → Indica el número (siendo 1 el primero) o identificador del marcador de posición al que se va a vincular el valor.
- ***&\$variable*** → indica el valor a vincular.
- ***\$data\_type*** → Indica el tipo de dato de la columna a la que va dirigido el valor. Los tipos posibles están definidos mediante las siguientes constantes.

***PDO::PARAM\_BOOL (integer)*** → Representa un tipo de dato booleano.

***PDO::PARAM\_NULL (integer)*** → Representa el tipo de dato NULL de SQL.

***PDO::PARAM\_INT (integer)*** → Representa el tipo de dato INTEGER de SQL .

***PDO::PARAM\_STR (integer)*** → Representa el tipo de dato CHAR, VARCHAR de SQL, u otro tipo de datos de cadena. **( Este es el tipo predeterminado si no se indica otro )**

***PDO::PARAM\_LOB (integer)*** → Representa el tipo de dato de objeto grande (LOB) de SQL.

***PDO::PARAM\_INPUT\_OUTPUT (integer)*** → Especifica que el parámetro es de entrada/salida (INOUT) para un procedimiento almacenado. Se debe realizar un OR a nivel de bit con un tipo de datos PDO::PARAM\_\* explícito.

- ***\$length*** → longitud del tipo de dato, en concreto; nº de caracteres máximos permitidos para la columna a la que va dirigido el valor.

## Programación Web en PHP

**Ejemplo:** El siguiente código es una modificación del ejemplo anterior en el que se muestran todas las cuentas bancarias cuyo saldo sea superior al indicado por el usuario en el formulario. (\*) En la parte HTML debe sustituirse el name de la caja de texto por el valor 'cantidad'.

```
<?php
if ( isset($_POST['cantidad'])) {
    $host = '192.168.9.180'; // IP del servidor
    $db = 'banco';           // Base de datos
    $port = 3306;           // Puerto
    $charset = 'utf8';       // Set de caracteres
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
    $user = 'alumno';        // Usuario
    $pass = 'cipsa';         // Contraseña
    // Opciones de configuracion
    $opt = [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false,
    ];
    // Instancia de PDO.
    $pdo = new PDO($dsn, $user, $pass, $opt);
    // Declaración sentencia parametrizada empleando
    $resultados = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
FROM Cuentas WHERE SALDO > :saldo");
    // Vinculación de variable $saldo.
    $resultados->bindParam(":saldo", $saldo, PDO::PARAM_INT);
    // Asignación de valor a la variable vinculada.
    $saldo = $_POST['cantidad'];
    // Ejecución del comando
    $resultados->execute();
}
?>
```

**Ejemplo 2:** El siguiente código es otra modificación del ejemplo anterior en la que se muestran todas las cuentas dadas de alta en una fecha posterior a la indicada por el usuario ( formato: dd/mm/yyyy) y con el estado de bloqueado indicado.

```
<?php
if ( !empty($_POST) ) {
    $host = '192.168.9.180'; // IP del servidor
    $db = 'banco';           // Base de datos
    $port = 3306;           // Puerto
    $charset = 'utf8';       // Set de caracteres
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
    $user = 'alumno';        // Usuario
    $pass = 'cipsa';         // Contraseña
    // Opciones de configuracion
    $opt = [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false,
    ];
    // Instancia de PDO.
    $pdo = new PDO($dsn, $user, $pass, $opt);
    // Declaración sentencia parametrizada empleando
    $resultados = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES FROM
Cuentas WHERE BLOQUEADA = :bloqueada AND APERTURA > :apertura");
    // Vinculación de parámetros de entrada
    $resultados->bindParam(":bloqueada", $bloqueada, PDO::PARAM_BOOL);
    $resultados->bindParam(":apertura", $apertura, PDO::PARAM_STR);
    // Obtiene el valor lógico del checkbox ( 1 - marcado / 0 - desmarcado )
    $bloqueada = isset($_POST['bloqueada'])?1:0;
    // Obtiene la fecha expresada en d/m/Y y la convierte a Y-m-d H:i:s para MySQL
    $fecha = DateTime::createFromFormat("d/m/Y", $_POST['apertura']);
    $apertura = $fecha->format('Y-m-d H:i:s');
    // Ejecución indicando valores para parámetros
    $resultados->execute();
}
?>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form method="post">
        Apertura d/m/a: <input type="text" name="apertura">
        Bloqueada: <input type="checkbox" name="bloqueada">
        <input type="submit" value="Obtener">
    </form>
    <table style='width: 100%'>
    <tr><td>NCUENTE</td><td>TITULAR</td><td>SALDO</td><td>INTERES</td>
    <?php
        if ( isset($resultados) ) {
            while ( $reg = $resultados->fetch() ) {
                ?>
                <tr>
                <td><?=$reg['NUMERO_CUENTA'] ?></td>
                <td><?=$reg['TITULAR'] ?></td>
                <td><?=$reg['SALDO'] ?></td>
                <td><?=$reg['INTERES'] ?></td>
                </tr>
            }
        }
    <?php
    </table>
</body>
</html>
```

En ambos casos, una vez ejecutado el método **bindParam()**, se sustituirá el marcador de posición :saldo por el valor de \$saldo cuando se llame al método **execute()**.

(\*) La vinculación de parámetros es equivalente a indicar los valores para los marcadores de posición en el método **execute()**. Existen sin embargo ocasiones en las que es necesario emplear la vinculación de parámetros de entrada. Esto sucede cuando se trabaja con columnas de tipo *BOOL*, *BIGINT*, o *procedimientos almacenados*.

### Acerca de las fechas

MySQL recibe las fechas en un formato "año-mes-día hora:minutos:segundos". Cualquier valor que se pase para una columna de tipo DATE, debe llevar la fecha expresada en el formato indicado. Sin embargo, la fecha indicada por el usuario la podemos recoger en nuestro formato "día/mes/año". Para convertir el formato de recogida en el de MySQL puede emplearse las siguientes funciones:

- Método estático **CreateFromFormat()** de la clase **Date** → Devuelve un objeto Date con la fecha indicada como parámetro en el formato dado.
- Método **format()** de la clase **Date** → Devuelve una cadena con la fecha expresada en el formato indicado.

### Acerca de los valores lógicos

Los valores lógicos se vinculan indicando el tipo **PDO::PARAM\_BOOL** indicando el valor 1 para cierto, y 0 para falso.

## Ejecución de comandos de modificación de valores.

Los comandos SQL de modificación de datos ( *INSERT*, *UPDATE*, *DELETE* ) se ejecutan del mismo modo que las consultas, con la diferencia que no devuelven un conjunto de resultados. En su caso, puede obtenerse el nº de registros modificador empleando el método *rowCount()* del objeto *PDOStatement*.

**Ejemplo:** El siguiente código actualiza el saldo de la cuenta con el nº de cuenta indicada aplicándole los intereses correspondientes si no está bloqueada. Finalmente muestra el saldo actualizado de la cuenta:

```
<?php
$host = '192.168.9.180'; // IP del servidor
$db = 'banco';           // Base de datos
$port = 3306;            // Puerto
$charset = 'utf8';       // Set de caracteres
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";

$user = 'alumno';        // Usuario
$pass = 'cipsa';         // Contraseña

// Opciones de configuracion
$opt = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES  => false,
];
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);

// Consulta parametrizada de actualizacion
$resultados = $pdo->prepare("UPDATE CUENTAS SET SALDO = SALDO + ( SALDO *
(INTERES/100) ) WHERE BLOQUEADA = 0 AND NUMERO_CUENTA = :cuenta");
$resultados->bindParam(":cuenta", $cuenta, PDO::PARAM_STR, 8);
$cuenta = 'AC129384';
$resultados->execute();

// Comprobacion de registros modificados
if ( $resultados->rowCount() == 1 ) {
    // Se modifico 1 registro -> Consulta para obtener saldo actualizado
    $consulta = $pdo->prepare("SELECT SALDO FROM CUENTAS WHERE NUMERO_CUENTA =
:cuenta");
    $consulta->bindParam(":cuenta", $cuenta, PDO::PARAM_STR, 8);
    $consulta->execute();
    // Obtención de saldo actualizado
    $registro = $consulta->fetch();
    echo "El nuevo saldo es: ".$registro['SALDO'];
} else {
    // Se modificó 0 registros -> No hay cuenta con el nº de cuenta indicado
    echo "La cuenta no existe";
}
?>
```

El método *rowCount()* devuelve un valor 1 si el nº de cuenta existe y se actualiza un registro, o 0 en caso contrario.

### Obtención del último valor autogenerado

Los campos de tipo autonumérico (**AUTO\_INCREMENT**) son aquellos cuyo valor es generado por la base de datos al insertar un nuevo registro. Este valor obtenerse una vez insertado el registro empleando la función **lastInsertId()** del objeto PDO:

**Ejemplo:** El siguiente código inserta un nuevo registro en la tabla movimientos y obtiene el valor del campo autonumérico **ID\_MOVIMIENTO**:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
// Declaración sentencia parametrizada empleando
$resultados = $pdo->prepare("INSERT INTO MOVIMIENTOS( CANTIDAD, CONCEPTO, CUENTA, FECHA
) VALUES ( :cantidad, :concepto, :cuenta, :fecha)");
// Vinculación de parámetros de entrada
$resultados->execute(
    [':cuenta' => 'AC12934', // NUMERO_CUENTA de cuenta asociada al movimiento
    ':cantidad' => 1200.40,
    ':concepto' => 'Pago impuesto de circulación',
    ':fecha' => (new DateTime())->format('Y-m-d H:i:s')]); // Fecha actual.
// Obtención de campo ID_MOVIMIENTO generado automáticamente.
$id_movimiento = $pdo->lastInsertId();
```

Al crear la **INSERT** no debe indicarse un valor para el campo ID MOVIMIENTO, puesto que es autonumérico. El valor para el campo **CUENTA** debe corresponderse con el valor del campo NUMERO CUENTA de algún registro de la tabla CUENTAS debido a la restricción de clave externa entre ambas tablas. En caso contrario se producirá un error de violación de clave externa.

```
[!] Fatal error: Uncaught PDOException: SQLSTATE[23000]: Integrity constraint violation: 1452 Cannot add or update a child row: a foreign key constraint fails ('banco'.movimientos, CONSTRAINT `movimientos_ibfk_1` FOREIGN KEY (`CUENTA`) REFERENCES `cuentas` (`NUMERO_CUENTA`)) in C:\xampp7\htdocs\prueba\database\move.php on line 29
```

## Recuperación de resultados

La recuperación de resultados puede realizarse de tres modos:

### Recuperación de múltiples registros de uno en uno

La recuperación de resultados se realiza mediante el método ***fetch()*** del objeto ***PDOStatement***:

```
public mixed PDOStatement::fetch ([ int $fetch_style  
                                [, int $cursor_orientation = PDO::FETCH_ORI_NEXT  
                                [, int $cursor_offset = 0 ]]] )
```

El modo en que este método devuelve los datos depende del valor dado al parámetro ***\$fetch\_style***. Los valores posibles están definidos por las siguientes constantes:

- ***PDO::FETCH\_NUM*** → Retorna los resultados como una matriz escalar.
- ***PDO::BOTH*** → Retorna una matriz combinada de ambos tipos. Cada campo es accesible tanto empleando un índice numérico ( siendo 0 el primer campo indicado en la SELECT ), como el propio nombre del campo.
- ***PDO::FETCH\_OBJ*** → Retorna una matriz de objetos anónimos donde cada atributo se corresponde con un campo del registro y sus nombres con los de las columnas.
- ***PDO::FETCH\_ASSOC*** → Retorna los resultados como una matriz asociativa indicando como claves los nombres de las columnas.

**Ejemplo:** Modo ***FETCH\_NUM***. Cada campo se recupera mediante un índice numérico siendo 0 el primer campo indicado en la ***SELECT***:

```
// Instancia de PDO.  
$pdo = new PDO($dsn, $user, $pass, $opt);  
$consulta = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES, BLOQUEADA,  
APERTURA FROM CUENTAS");  
while($reg = $consulta->fetch(PDO::FETCH_NUM)) {  
    // Obtencion nº cuenta  
    echo "NUMERO_CUENTA: $reg[0]<br>";  
    // Obtencion titular  
    echo "TITULAR: $reg[1]<br>";  
    // Volcado datos registro  
    var_dump($reg);  
}
```

```
NUMERO_CUENTA: AC129384  
TITULAR: ROGER PETROV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
array (size=6)  
  0 => string 'AC129384' (length=8)  
  1 => string 'ROGER PETROV' (length=12)  
  2 => string '8751.65' (length=7)  
  3 => string '5.00' (length=4)  
  4 => int 0  
  5 => string '2017-03-22' (length=10)  
NUMERO_CUENTA: AC439586  
TITULAR: YURI PISKUNOV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
array (size=6)  
  0 => string 'AC439586' (length=8)  
  1 => string 'YURI PISKUNOV' (length=13)  
  2 => string '550.00' (length=6)  
  3 => string '0.00' (length=4)  
  4 => int 0  
  5 => string '2017-03-21' (length=10)
```

**Ejemplo:** Modo ***FETCH\_ASSOC***. Cada campo se recupera mediante el nombre del campo indicado en la consulta ***SELECT***:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES, BLOQUEADA,
APERTURA FROM CUENTAS");
while($reg = $consulta->fetch(PDO::FETCH_ASSOC)) {
    echo "NUMERO_CUENTA: ".$reg['NUMERO_CUENTA']."<br>";
    echo "TITULAR: ".$reg['TITULAR']."<br>";
    var_dump($reg);
}
```

NUMERO\_CUENTA: AC129384  
TITULAR: ROGER PETROV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
array (size=6)  
'NUMERO\_CUENTA' => string 'AC129384' (length=8)  
'TITULAR' => string 'ROGER PETROV' (length=12)  
'SALDO' => string '8751.65' (length=7)  
'INTERES' => string '5.00' (length=4)  
'BLOQUEADA' => int 0  
'APERTURA' => string '2017-03-22' (length=10)  
NUMERO\_CUENTA: AC439586  
TITULAR: YURI PISKUNOV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
array (size=6)  
'NUMERO\_CUENTA' => string 'AC439586' (length=8)  
'TITULAR' => string 'YURI PISKUNOV' (length=13)  
'SALDO' => string '550.00' (length=6)  
'INTERES' => string '0.00' (length=4)  
'BLOQUEADA' => int 0  
'APERTURA' => string '2017-03-21' (length=10)

**Ejemplo:** Modo ***FETCH\_OBJ***. Cada campo se recupera como una propiedad del objeto devuelto por el método ***fetch()***:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES, BLOQUEADA,
APERTURA FROM CUENTAS");
while($reg = $consulta->fetch(PDO::FETCH_OBJ)) {
    // Obtencion nº cuenta
    echo "NUMERO_CUENTA: $reg->NUMERO_CUENTA<br>";
    // Obtencion titular
    echo "TITULAR: $reg->TITULAR<br>";
    // Volcado datos registro
    var_dump($reg);
}
```

NUMERO\_CUENTA: AC129384  
TITULAR: ROGER PETROV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
object(stdClass) [3]  
public 'NUMERO\_CUENTA' => string 'AC129384' (length=8)  
public 'TITULAR' => string 'ROGER PETROV' (length=12)  
public 'SALDO' => string '8751.65' (length=7)  
public 'INTERES' => string '5.00' (length=4)  
public 'BLOQUEADA' => int 0  
public 'APERTURA' => string '2017-03-22' (length=10)  
NUMERO\_CUENTA: AC439586  
TITULAR: YURI PISKUNOV  
C:\xampp7\htdocs\prueba\database\update.php:27:  
object(stdClass) [4]  
public 'NUMERO\_CUENTA' => string 'AC439586' (length=8)  
public 'TITULAR' => string 'YURI PISKUNOV' (length=13)  
public 'SALDO' => string '550.00' (length=6)  
public 'INTERES' => string '0.00' (length=4)  
public 'BLOQUEADA' => int 0  
public 'APERTURA' => string '2017-03-21' (length=10)

El estilo predeterminado es **PDO::BOTH** salvo que se declare un estilo predeterminado al instanciar el objeto PDO en el parámetro **PDO::ATTR\_DEFAULT\_FETCH\_MODE** como parte de las opciones de configuración:

**Ejemplo:** El siguiente código muestra la declaración del modo **PDO::FETCH\_ASSOC** como estilo predeterminado para el método **fetch()**.

```
<?php
$host = '192.168.9.180';      // IP del servidor
$db   = 'banco';              // Base de datos
$port = 3306;                 // Puerto
$charset = 'utf8';            // Set de caracteres
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";

$user = 'alumno';             // Usuario
$pass = 'cipsa';              // Contraseña

// Opciones de configuracion
$opt = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES  => false,
];
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
```

### Recuperación de una única columna

También es posible recuperar una única columna empleando el método **fetchColumn()** del objeto **PDOStatement**. Este método funciona igual que el método **fetch()**, pero con la diferencia de que únicamente retorna el valor de la columna indicada:

```
public mixed PDOStatement::fetchColumn ( [ int $column_number = 0 ] )
```

El método recibe como argumento un índice que indica la columna que se desea devolver siendo 0 la primera ( o única ) indicada en la consulta. Si no se indica ningún índice, devuelve el primer campo. Si no existe ningún registro retorna FALSO:

Este método es especialmente útil para recuperar el valor devuelto por consultas escalares que retornan un único registro con un único campo:

**Ejemplo 1:** Consulta que devuelve el sumatorio de los saldos de todas las cuentas registradas:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT SUM(SALDO) FROM CUENTAS");
// Obtiene el primer campo del siguiente registro.
$sumatorio = $consulta->fetchColumn();
echo "Sumatorio de las cuenta: $sumatorio €<br>";
```

**Ejemplo 2:** Consulta que retorna el nº de cuentas con saldo superior a 1000€:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->prepare("SELECT COUNT(*) FROM CUENTAS WHERE SALDO > :saldo");
$consulta->execute([':saldo' => 1000]);
// Obtencion resultado de la consulta escalar
$n_cuentas = $consulta->fetchColumn();
echo "Numero de cuentas: ".$n_cuentas;
```



## Recuperación de conjunto de resultados completo

La recuperación del conjunto de resultados al completo permite obtener todos los registros de una sola vez ( sin bucles ) en vez de secuencialmente. Para ello debe emplearse el método ***fetchAll()*** del objeto ***PDOStatement***.

```
public array PDOStatement::fetchAll ([ int $fetch_style
                                [, mixed $fetch_argument
                                [, array $ctor_args = array() ]]] )
```

El parámetro ***\$fetch\_style*** determina el modo en que se devuelven los datos.

Este método es especialmente útil cuando los datos deben manejarse en conjunto en vez de uno en uno, como en el caso de las plantillas; que requieren recibir todos los datos a mostrar para generar la página web resultante.

### Recuperación en una matriz de matrices.

De manera predeterminada, los datos de todos los registros son devueltos en la forma de una matriz escalar, en cada una de cuyas posiciones se almacena una con los datos de los campos de cada registro según la constante indicada: ***PDO::FETCH\_NUM***, ***PDO::FETCH\_BOTH***, ***PDO::FETCH\_ASSOC***, o ***PDO::FETCH\_OBJ***.

Si no se indica ningún valor, se asume el valor indicado en la opción ***PDO::ATTR\_DEFAULT\_FETCH\_MODE*** del objeto ***PDO***, o ***PDO::FETCH\_BOTH*** en su defecto.

**Ejemplo:** El siguiente código devuelve una matriz de objetos con los datos de todas las cuentas registradas:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES, BLOQUEADA,
APERTURA FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_OBJ);
var_dump($resultados);

array (size=2)
  0 =>
    object(stdClass) [3]
      public 'NUMERO_CUENTA' => string 'AC129384' (length=8)
      public 'TITULAR' => string 'ROGER PETROV' (length=12)
      public 'SALDO' => string '8751.65' (length=7)
      public 'INTERES' => string '5.00' (length=4)
      public 'BLOQUEADA' => int 0
      public 'APERTURA' => string '2017-03-22' (length=10)
  1 =>
    object(stdClass) [4]
      public 'NUMERO_CUENTA' => string 'AC439586' (length=8)
      public 'TITULAR' => string 'YURI PISKUNOV' (length=13)
      public 'SALDO' => string '550.00' (length=6)
      public 'INTERES' => string '0.00' (length=4)
      public 'BLOQUEADA' => int 0
      public 'APERTURA' => string '2017-03-21' (length=10)
```

## Recuperación en una matriz de valores

Cuando se realiza una consulta que únicamente devuelve un campo de cada registro, éstos pueden obtenerse en una matriz escalar con el valor de cada registro. Para ello debe indicarse la constante **PDO::FETCH\_COLUMN** como valor para el parámetro *\$fetch\_style*:

**Ejemplo:** La siguiente consulta devuelve una matriz con los nº de cuenta de todas las cuentas registradas:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT NUMERO_CUENTA FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_COLUMN);
var_dump($resultados);

array (size=2)
  0 => string 'AC129384' (length=8)
  1 => string 'AC439586' (length=8)
```

## Recuperación en una matriz indexada por clave primaria

Este modo permite recuperar los registros de una consulta en una matriz cuyos índices se correspondan con el valor la clave primaria. Para ello debe indicarse la constante **PDO::FETCH\_UNIQUE** como valor para el parámetro *\$fetch\_style*.

**Ejemplo:** La siguiente consulta devuelve todas las cuentas registradas en una matriz asociativa donde la clave es la clave primaria de la tabla (NUMERO\_CUENTA), y cada cuenta se almacena como una matriz asociativa con el nombre de cada columna:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES, BLOQUEADA, APERTURA FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_UNIQUE);
var_dump($resultados);

array (size=2)
  'AC129384' =>
    array (size=5)
      'TITULAR' => string 'ROGER PETROV' (length=12)
      'SALDO' => string '8751.65' (length=7)
      'INTERES' => string '5.00' (length=4)
      'BLOQUEADA' => int 0
      'APERTURA' => string '2017-03-22' (length=10)
  'AC439586' =>
    array (size=5)
      'TITULAR' => string 'YURI PISKUNOV' (length=13)
      'SALDO' => string '550.00' (length=6)
      'INTERES' => string '0.00' (length=4)
      'BLOQUEADA' => int 0
      'APERTURA' => string '2017-03-21' (length=10)
```

## Recuperación como objetos de una clase existente.

Este modo permite obtener los datos de cada registro devuelto como un objeto de una determinada clase ya definida (en vez de una anónima, como al indicar **PDO::FETCH\_OBJ**). Para ello debe indicarse la constante **PDO::FETCH\_CLASS**, para el parámetro *\$fetch\_style*, seguido del nombre de la clase.

**Ejemplo:** El siguiente código muestra la recuperación de todas las cuentas registradas como una matriz asociativa con los números de cuenta como clave y de objetos de la clase **cuenta** con los datos de cada registro como valor.

```
class cuenta {
    private $numero_cuenta;
    private $titular;
    private $saldo;
    private $interes;
    private $bloqueada;
    private $apertura;

    private $beneficio;    // Atributo calculado en el constructor

    public function __construct() {
        // Cálculo del valor del atributo en funcion de saldo e interes.
        $this->beneficio = $this->saldo * ($this->interes/100.0);
    }
}

// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT numero_cuenta, titular, saldo, interes, bloqueada,
apertura FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_CLASS|PDO::FETCH_UNIQUE, "cuenta");
var_dump($resultados);

array (size=4)
'AC129384' =>
  object(cuenta) [3]
    private 'numero_cuenta' => null
    private 'titular' => string 'ROGER PETROV' (length=12)
    private 'saldo' => string '8751.65' (length=7)
    private 'interes' => string '5.00' (length=4)
    private 'bloqueada' => int 0
    private 'apertura' => string '2017-03-22' (length=10)
    private 'beneficio' => float 437.5825
'AC439586' =>
  object(cuenta) [4]
    private 'numero_cuenta' => null
    private 'titular' => string 'YURI PISKUNOV' (length=13)
    private 'saldo' => string '550.00' (length=6)
    private 'interes' => string '0.02' (length=4)
    private 'bloqueada' => int 0
    private 'apertura' => string '2017-03-21' (length=10)
    private 'beneficio' => float 0.11
'AC490385' =>
  object(cuenta) [5]
    private 'numero_cuenta' => null
    private 'titular' => string 'ROGER PETROV' (length=12)
    private 'saldo' => string '550.45' (length=6)
    private 'interes' => string '0.05' (length=4)
    private 'bloqueada' => int 0
    private 'apertura' => string '2017-03-05' (length=10)
    private 'beneficio' => float 0.275225
'AC594045' =>
  object(cuenta) [6]
    private 'numero_cuenta' => null
    private 'titular' => string 'ROGER PETROV' (length=12)
    private 'saldo' => string '120.40' (length=6)
    private 'interes' => string '0.04' (length=4)
    private 'bloqueada' => int 0
    private 'apertura' => string '2017-03-20' (length=10)
    private 'beneficio' => float 0.04816
```

### Consideraciones importantes:

El método ***fetchAll()*** crea un objeto de la clase indicada por cada registro y añade los valores de sus campos a los atributos de la clase con las siguientes normas:

- Si no se encuentra un atributo con el nombre del campo → Se invoca al método mágico ***\_\_set()***. Si tampoco está definido el método mágico ***\_\_set()*** → Se añade un atributo con el nombre del campo.
- En caso de declarar el método ***\_\_construct()***, debe ser no parametrizado y se ejecuta para cada objeto una vez inicializados los atributos.

### Recuperación agrupada por el valor de un campo

La recuperación agrupada permite obtener los registros agrupados en una matriz por el valor de un determinado campo. Para ello debe indicarse la constante ***PDO::FETCH\_GROUP*** como valor para el parámetro *\$fetch\_style*:

**Ejemplo:** El siguiente código obtiene una matriz asociativa con los nombres de los clientes del banco (campo *titular*), y por cada uno, una matriz con los datos de cada una de sus cuentas. Cada cuenta se almacena a su vez como una matriz asociativa con los nombres de los campos.

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT TITULAR, NUMERO_CUENTA, SALDO, INTERES, BLOQUEADA,
APERTURA FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_GROUP);
var_dump($resultados);

array (size=2)
  'ROGER PETROV' =>
    array (size=3)
      0 =>
        array (size=5)
          'numero cuenta' => string 'AC129384' (length=8)
          'saldo' => string '8751.65' (length=7)
          'interes' => string '5.00' (length=4)
          'bloqueada' => int 0
          'apertura' => string '2017-03-22' (length=10)
      1 =>
        array (size=5)
          'numero cuenta' => string 'AC490385' (length=8)
          'saldo' => string '550.45' (length=6)
          'interes' => string '0.05' (length=4)
          'bloqueada' => int 0
          'apertura' => string '2017-03-05' (length=10)
      2 =>
        array (size=5)
          'numero cuenta' => string 'AC594045' (length=8)
          'saldo' => string '120.40' (length=6)
          'interes' => string '0.04' (length=4)
          'bloqueada' => int 0
          'apertura' => string '2017-03-20' (length=10)
  'YURI PISKUNOV' =>
    array (size=1)
      0 =>
        array (size=5)
          'numero cuenta' => string 'AC439586' (length=8)
          'saldo' => string '550.00' (length=6)
          'interes' => string '0.02' (length=4)
          'bloqueada' => int 0
          'apertura' => string '2017-03-21' (length=10)
```

En este caso, por los resultados se ve que el titular *"ROGER PETROV"* cuenta con tres cuentas bancarias.

## Recuperación de conjuntos de resultados personalizados

Esta opción permite obtener los datos organizados como deseemos empleando para ello una función que se ejecuta por cada registro y debe retornar los datos del modo deseado. Para ello debe indicarse la constante **PDO::FETCH\_FUNC** como valor para el parámetro *\$fetch\_style* acompañado de la función.

**Ejemplo 1:** El siguiente código devuelve una matriz escalar con los siguientes datos para cada cuenta registrada en la tabla cuentas como una matriz asociativa:

- Id → Concatena el nº de cuenta y el titular
- Saldo → Saldo de la cuenta
- Interés → Interés de la cuenta

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT numero_cuenta, titular, saldo, interes FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_FUNC,
    function ($nc, $t, $s, $i) {
        return [ 'id' => $nc . ' > ' . $t, 'saldo' => $s, 'interes' => $i];
    }
);
var_dump($resultados);
```

```
array (size=4)
  0 =>
    array (size=3)
      'id' => string 'AC129384 > ROGER PETROV' (length=23)
      'saldo' => string '8751.65' (length=7)
      'interes' => string '5.00' (length=4)
  1 =>
    array (size=3)
      'id' => string 'AC439586 > YURI PISKUNOV' (length=24)
      'saldo' => string '550.00' (length=6)
      'interes' => string '0.02' (length=4)
  2 =>
    array (size=3)
      'id' => string 'AC490385 > ROGER PETROV' (length=23)
      'saldo' => string '550.45' (length=6)
      'interes' => string '0.05' (length=4)
  3 =>
    array (size=3)
      'id' => string 'AC594045 > ROGER PETROV' (length=23)
      'saldo' => string '120.40' (length=6)
      'interes' => string '0.04' (length=4)
```

La función anónima empleada recibe un parámetro para cada uno de los campos devueltos por la consulta, y retorna como resultado una matriz asociativa que es lo que se obtiene como resultado en la matriz de resultados.

En caso de querer que la matriz de resultados contenga los datos de cada registro en un objeto en vez de en una matriz asociativa; el código de la función podría ser:

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT numero_cuenta, titular, saldo, interes FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_FUNC,
    function ($nc, $t, $s, $i) {
        $obj = new stdClass();
        $obj->id = $nc . ' > ' . $t;
        $obj->saldo = $s;
        $obj->interes = $i;
        return $obj;
    }
);
var_dump($resultados);
```

**Ejemplo 2:** El siguiente código devuelve una matriz asociativa organizada por nº de cuenta y la fecha de apertura en codificación europea (día/mes/año).

En este caso la función anónima indicada sólo recibe un parámetro, ya que el primer campo devuelto (*numero\_cuenta*) es empleado para el agrupamiento debido a la presencia de la constante **PDO::FETCH\_UNIQUE**.

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);
$consulta = $pdo->query("SELECT numero_cuenta, apertura FROM CUENTAS");
$resultados = $consulta->fetchAll(PDO::FETCH_UNIQUE|PDO::FETCH_FUNC,
    function ($apertura) {
        $date = DateTime::createFromFormat('Y-m-d H:i:s', $apertura);
        return $date->format('d/m/Y h:i a');
    });
var_dump($resultados);

array (size=4)
  'AC129384' => string '22/03/2017 12:00 am' (length=19)
  'AC439586' => string '21/03/2017 12:00 am' (length=19)
  'AC490385' => string '05/03/2017 12:00 am' (length=19)
  'AC594045' => string '20/03/2017 12:00 am' (length=19)
```

### Consultas con/sin buffer

Cuando se realiza una consulta mediante un comando SELECT el conjunto de resultados generado puede manejarse de dos modos:

- **Modo buffer** → El conjunto de resultados es volcado a la memoria del servidor. Esto es así tanto si accedamos a los registros de uno en uno mediante el método **fetch()**, o a todos a la vez mediante **fetchAll()**.
- **Modo no buffer** → Sólo se almacena en la memoria del servidor los datos de cada registro según se van recuperando de uno en uno mediante el método **fetch()**.

El uso del *modo buffer* impone ciertas limitaciones:

- No es posible emplear el método **rowCount()** para conocer el nº de registros que componen el conjunto de resultados, ya que se recuperan de uno en uno.
- No es posible realizar otra consulta mientras se están obteniendo los resultados de una anterior a través del mismo objeto PDO.
- Los registros sólo pueden recuperarse hacia delante. ( del primero al último ).

El uso del *modo buffer* sacrifica memoria pero mejora el rendimiento al tenerse todos los registros del conjunto de resultados en la memoria del servidor en vez de tener que obtenerlos desde el servidor de base de dato, lo cual es más lento.

### Recomendación.

Se desaconseja el uso del modo buffer al realizar consultas que devuelvan conjuntos de resultados compuestos por muchos registros, esto es; más de los que pueden mostrarse cómodamente en una web al usuario. De lo contrario, podría rebasarse el límite de memoria permitida para la ejecución de los scripts en el servidor y provocar un error durante la ejecución.

**(\*) El límite de memoria del servidor que un script puede emplear durante su ejecución está definido mediante la directiva `memory_limit` en el archivo de configuración de PHP (`php.ini`). El valor predeterminado es 128Mbytes.**

La configuración del modo buffer y sin buffer se realiza mediante el método **`setAttribute()`** del objeto **`PDO`** indicando la opción de configuración **`PDO::MYSQL_ATTR_USER_BUFFERED_QUERY`**, donde los posibles valores son **`TRUE`** / **`FALSE`**.

**Ejemplo:** Consulta empleando buffer para el conjunto de resultados.

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);


$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);


$consulta = $pdo->query("SELECT numero_cuenta, titular, saldo, interes FROM CUENTAS");
while( $reg = $consulta->fetch() ) {
    var_dump($reg);
}
echo memory_get_peak_usage(); // Indica la memoria máxima empleada
```

**Ejemplo:** Consulta sin emplear buffer para el conjunto de resultados.

```
// Instancia de PDO.
$pdo = new PDO($dsn, $user, $pass, $opt);


$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);


$consulta = $pdo->query("SELECT numero_cuenta, titular, saldo, interes FROM CUENTAS");
while( $reg = $consulta->fetch() ) {
    var_dump($reg);
}
echo memory_get_peak_usage(); // Indica la memoria máxima empleada
```

**Ejemplo:** Consulta limitada sobre tabla movimientos de un individuo.

## Manejo de errores

Los errores en el manejo de una base de datos pueden darse principalmente en dos momentos:

- Al instanciar el objeto PDO que representa la conexión a la base de datos.
- Al ejecutar un comando SQL.

Los errores pueden capturarse mediante excepciones (recomendado) para lo cual deben indicarse en las opciones de conexión la opción **PDO::ATTR\_ERRMODE** asignando el valor **PDO::ERRMODE\_EXCEPTION**. Esta configuración permite que las excepciones generadas contengan información adicional sobre el error mediante los métodos **getCode()** y **getMessage()**.

**Ejemplo:** El siguiente código muestra una consulta a la base de datos incluyendo captura de excepciones por si se producen errores en el momento de la conexión y la realización de la consulta:

```
$host = '192.168.9.180'; // IP del servidor
$db = 'banco'; // Base de datos
$port = 3306; // Puerto
$charset = 'utf8'; // Set de caracteres
$dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
$user = 'alumno'; // Usuario
$pass = 'cipsa'; // Contraseña
// Opciones de configuracion
$opt = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false
];
// Establecimiento de la conexion.
try {
    // Obtencion objeto conexion
    $pdo = new PDO($dsn, $user, $pass, $opt);
} catch (Exception $e) {
    // Error al establecer conexion
    echo "ERROR DE CONEXION: <br/>";
    echo "CODIGO: " . $e->getCode() . "<br/>";
    echo "CAUSA: " . $e->getMessage() . "<br/>";
    die();
}
// Ejecucion de consulta
try {
    // Declaración y ejecución de consulta
    $sentencia = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES
    FROM CUENTAS WHERE TITULAR LIKE :titular");
    $sentencia->execute([':titular' => 'ROGER PETROV']);
} catch (Exception $e) {
    // Error de ejecución de consulta
    echo "ERROR EN CONSULTA: <br/>";
    echo "CODIGO: " . $e->getCode() . "<br/>";
    echo "CAUSA: " . $e->getMessage() . "<br/>";
    die();
}
// Visualizacion de resultados
while( $reg = $sentencia->fetch() ) {
    var_dump($reg);
}
// Cerrado de consulta
unset($sentencia);
// Cerrado de conexion
unset($pdo);
```



En el ejemplo anterior, hay dos bloques **try\_catch**:

El primero captura posibles excepciones al establecer la conexión con la base de datos creando el objeto PDO. Estas pueden darse por errores en los parámetros de conexión: (IP del servidor, puerto, nombre de la base de datos, usuario y contraseña):

```
try {  
    // Obtencion objeto conexion  
    $pdo = new PDO($dsn, $user, $pass, $opt);  
} catch ( Exception $e ) {  
    // Error al establecer conexion  
    echo "ERROR DE CONEXION: <br/>";  
    echo "CODIGO: " . $e->getCode() . "<br/>";  
    echo "CAUSA: " . $e->getMessage() . "<br/>";  
    die();  
}
```

El segundo captura excepciones debidas a la ejecución de la consulta. Estas pueden darse al obtener el objeto *\$sentencia* mediante *prepare()* si el comando SQL es incorrecto; o al ejecutarse mediante *execute()* si no asignan valores adecuados para todos los parámetros de la consulta:

```
try {  
    // Declaración y ejecución de consulta  
    $sentencia = $pdo->prepare("SELECT NUMERO_CUENTA, TITULAR, SALDO, INTERES  
FROM CUENTAS WHERE TITULAR LIKE :titular");  
    $sentencia->execute([':titular' => 'ROGER PETROV']);  
} catch ( Exception $e ) {  
    // Error de ejecución de consulta  
    echo "ERROR EN CONSULTA: <br/>";  
    echo "CODIGO: " . $e->getCode() . "<br/>";  
    echo "CAUSA: " . $e->getMessage() . "<br/>";  
    die();  
}
```

## Transacciones

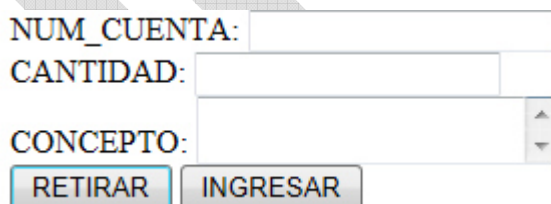
Las transacciones son un mecanismo que permite ejecutar múltiples sentencias SQL (normalmente de modificación de datos), de modo que los cambios no se registran definitivamente en la base de datos hasta que todas se han ejecutado correctamente.

Si por el contrario una de las sentencias falla pueden deshacerse los cambios realizados por todas las anteriores devolviendo la base de datos al estado inicial.

Para ello se emplean tres métodos de la clase PDO:

- ***beginTransaction()*** → Este método inicia la transacción. Retorna un valor lógico TRUE si la transacción se inicia correctamente y FALSO en caso de error. Si se habilita el lanzamiento de excepciones (RECOMENDADO), el método lanza una excepción si ya hay una transacción en curso. Todos los cambios realizados por las consultas ejecutadas tras la llamada a este método serán temporales hasta que se confirme o cancele la transacción mediante los métodos siguientes:
- ***commit()*** → Confirma la transacción en curso haciendo permanentes todos los cambios de las consultas ejecutadas desde el inicio de la transacción.
- ***rollback()*** → Cancela la transacción en curso deshaciendo todos los cambios realizados por las consultas ejecutadas desde el inicio de la transacción.

**Ejemplo:** El siguiente código muestra un formulario en el que puede introducirse un número de cuenta, una cantidad y un concepto para registrar un movimiento sobre la cuenta indicada.



Formulario de registro de movimiento de cuenta. Incluye tres campos de entrada: NUM\_CUENTA, CANTIDAD y CONCEPTO. Debajo de los campos hay dos botones: RETIRAR y INGRESAR.

La operación implica dos modificaciones en la base de datos.

- Alta del movimiento en la tabla de movimientos.
- Modificación del saldo en la cuenta.

*Ambas operaciones deben realizarse correctamente, tal que si no puede darse de alta el movimiento, no debe modificarse la cuenta. Igualmente, si no puede modificarse la cuenta no debe quedar registrado el movimiento.*

Para asegurar la realización de ambas operaciones o de ninguna si hubiese un fallo empleamos una transacción.

## Programación Web en PHP

```
<?php

if ( !empty($_POST) ) {
    // Obtencion de valores del formulario
    $cuenta = filter_input(INPUT_POST, "ncuenta");
    $cantidad = filter_input(INPUT_POST, "cantidad");
    $concepto = filter_input(INPUT_POST, "concepto");
    if ( array_key_exists("RETIRAR", $_POST) ) {
        $cantidad *= -1;
    }

    $host = '192.168.9.180'; // IP del servidor
    $db = 'banco'; // Base de datos
    $port = 3306; // Puerto
    $charset = 'utf8'; // Set de caracteres
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset;port=$port";
    $user = 'alumno'; // Usuario
    $pass = 'cipsa'; // Contraseña
    // Opciones de configuracion
    $opt = [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES => false ];

    try {
        $pdo = new PDO($dsn, $user, $pass, $opt);
        // ----- Inicio de transacción
        $pdo->beginTransaction();
        // Registro del movimiento
        $sentencia = $pdo->prepare("INSERT INTO MOVIMIENTOS( CANTIDAD, CONCEPTO, CUENTA, FECHA ) VALUES ( :cantidad, :concepto, :cuenta, :fecha)");
        $sentencia->execute([':cuenta' => $cuenta,
            ':cantidad' => $cantidad,
            ':concepto' => $concepto,
            ':fecha' => (new Datetime())->format('Y-m-d H:i:s')]);
        // Actualizacion de la cuenta
        $sentencia= $pdo->prepare("UPDATE CUENTAS SET SALDO = SALDO + :saldo WHERE NUMERO_CUENTA = :cuenta");
        $sentencia->execute([':saldo' => $cantidad,
            ':cuenta' => $cuenta]);
        // ----- Confirmación de transacción
        $pdo->commit();
    } catch (Exception $ex) {
        echo "Error al realizar la operación: ".$ex->getMessage();
        // ----- Cancelación de transacción
        $pdo->rollBack();
        die();
    }

    $sentencia = $pdo->prepare("SELECT SALDO FROM CUENTAS WHERE NUMERO_CUENTA = :cuenta");
    $sentencia->execute([':cuenta'=>$cuenta]);
    $nuevo_saldo = $sentencia->fetchColumn();
}

?>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form method="post">
        NUM_CUENTA: <input type="text" name="ncuenta"><br/>
        CANTIDAD: <input type="numeric" name="cantidad"><br/>
        CONCEPTO: <textarea name="concepto"></textarea><br />
        <input type="submit" name="RETIRAR" value="RETIRAR">
        <input type="submit" name="RETIRAR" value="INGRESAR">
    </form>

    <?php if ( isset($nuevo_saldo)) { ?>
        El nuevo saldo es: <?= $nuevo_saldo; ?>
    <?php } ?>
</body>
</html>
```

En el ejemplo anterior se inicia la transacción antes de realizar las dos operaciones llamando al método ***beginTransaction()***. A continuación se realizan las dos operaciones una detrás de la otra.

Si todo funciona correctamente y no se produce ninguna excepción se ejecuta el método ***commit()*** que guarda definitivamente. En caso de error se ejecuta el bloque ***catch()*** y se deshacen las operaciones mediante el método ***rollback()***.

```
try {
    $pdo = new PDO($dsn, $user, $pass, $opt);
    // ----- Inicio de transacción
    $pdo->beginTransaction();
    // Registro del movimiento
    $sentencia = $pdo->prepare("INSERT INTO MOVIMIENTOS( CANTIDAD, CONCEPTO,
CUENTA, FECHA ) VALUES ( :cantidad, :concepto, :cuenta, :fecha)");
    $sentencia->execute([':cuenta' => $cuenta,
        ':cantidad' => $cantidad,
        ':concepto' => $concepto,
        ':fecha' => (new Datetime())->format('Y-m-d H:i:s')]);
    // Actualización de la cuenta
    $sentencia= $pdo->prepare("UPDATE CUENTAS SET SALDO = SALDO + :saldo WHERE
NUMERO_CUENTA = :cuenta");
    $sentencia->execute([':saldo' => $cantidad,
        ':cuenta' => $cuenta]);
    // ----- Confirmación de transacción
    $pdo->commit();
} catch( Exception $ex) {
    echo "Error al realizar la operación: ".$ex->getMessage();
    // ----- Cancelación de transacción
    $pdo->rollback();
    die();
}
```

## Un wrapper sencillo para PDO

Un wrapper es básicamente una clase que envuelve un objeto para gestionar su uso simplificando el código de manejo del mismo. En el caso de un wrapper para PDO se trata de una clase empleada para acceder a los datos utilizando menos código.

La siguiente clase encapsula el acceso a una base de datos empleando PDO de manera simple pero eficaz. Puede emplearse en cualquier aplicación web para simplificar las operaciones de acceso a la base de datos.

```
<?php
// IP del servidor
define('DB_HOST', '192.168.9.180');
// Identificador de la base de datos
define('DB_NAME', 'banco');
// Identificador de usuario de acceso al servidor
define('DB_USER', 'alumno');
// Contraseña de usuario de acceso al servidor
define('DB_PASS', 'cipsa');
// Codificación de caracteres empleada
define('DB_CHAR', 'utf8');

class DB
{
    // Atributo estático que almacena único objeto PDO instanciado.
    protected static $instance = null;
    protected function __construct() {}
    protected function __clone() {}

    // Método de obtención de objeto PDO
    public static function instance()
    {
        // Si no existe
        if (self::$instance === null)
        {
            // Se crea
            $opt = array(
                PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                PDO::ATTR_EMULATE_PREPARES   => FALSE,
            );
            $dsn = 'mysql:host='.DB_HOST.';dbname='.DB_NAME.';charset='.DB_CHAR;
            self::$instance = new PDO($dsn, DB_USER, DB_PASS, $opt);
        }
        // Retorno de instancia PDO creada/existente.
        return self::$instance;
    }

    // Método estático para invocación de métodos de objeto PDO
    public static function __callStatic($method, $args)
    {
        // Ejecuta el método estático invocado con los argumentos dados
        // sobre el objeto PDO existente.
        return call_user_func_array(array(self::instance(), $method), $args);
    }

    // Método de ejecución de consulta parametrizada
    public static function run($sql, $args = [])
    {
        $stmt = self::instance()->prepare($sql);
        $stmt->execute($args);
        // Retorno de objeto PDOStatement.
        return $stmt;
    }
}
```

[https://phpdelusions.net/pdo/pdo\\_wrapper](https://phpdelusions.net/pdo/pdo_wrapper)

## Cualidades y manejo.

Las constantes declaradas **DB\_HOST**, **DB\_NAME**, y **DB\_CHAR** almacenan la dirección del servidor, el nombre de la base de datos y la codificación de caracteres deseada. **DB\_USER** y **DB\_PASS** representan el nombre de usuario y clave de la cuenta de acceso al servidor de base de datos.

### El método estático *instance()*.

La clase DB encapsula una instancia de PDO única. El método **instance()** sólo permite obtenerse un objeto PDO (*una conexión*). La primera vez que se invoca crea un objeto PDO, lo almacena en la variable estática *\$instance* y lo retorna. Si se vuelve a invocar más veces retorna la instancia ya existente en vez de crear una nueva.

### El método estático *run()*.

El método **run()** permite realizar una consulta parametrizada mediante una sola instrucción recibiendo como argumentos la consulta y la matriz de parámetros. El método encapsula las sentencias **prepare()** y **execute()** contra el objeto PDO almacenado en *\$instance* y retorna el objeto **PDOStatement** que da acceso al conjunto de resultados.

**Ejemplo:** El siguiente código obtiene una matriz de objetos anónimos con el nº de cuenta, titular y saldo de todas cuentas registradas en la base de datos.

```
// Realización de la consulta
$resultados = DB::run(
    'SELECT NUMERO_CUENTA, TITULAR, SALDO FROM CUENTAS WHERE TITULAR LIKE :titular',
    [':titular' => $titular]);
// Obtención de resultados
$cuentas = $resultados->fetchAll(PDO::FETCH_OBJ);
var_dump($cuentas);
array (size=3)
  0 =>
    object(stdClass) [3]
      public 'NUMERO_CUENTA' => string 'AC129384' (length=8)
      public 'TITULAR' => string 'ROGER PETROV' (length=12)
      public 'SALDO' => string '8751.65' (length=7)
      public 'INTERES' => string '5.00' (length=4)
  1 =>
    object(stdClass) [4]
      public 'NUMERO_CUENTA' => string 'AC490385' (length=8)
      public 'TITULAR' => string 'ROGER PETROV' (length=12)
      public 'SALDO' => string '550.45' (length=6)
      public 'INTERES' => string '0.05' (length=4)
  2 =>
    object(stdClass) [5]
      public 'NUMERO_CUENTA' => string 'AC594045' (length=8)
      public 'TITULAR' => string 'ROGER PETROV' (length=12)
      public 'SALDO' => string '120.40' (length=6)
      public 'INTERES' => string '0.04' (length=4)
```

## Practica

Se pide crear una aplicación de gestión bancaria empleando la base de datos 'banco' implementada anteriormente que conste de las siguientes páginas.

### Página principal ( *home.php* )

Esta página consta de una lista de operaciones y hipervínculos a las páginas correspondientes de cada opción:

- Alta de cuenta → *alta.php*
- Listado de cuenta → *listado.php*
- Movimiento → *movimiento.php*
- Traspaso → *traspaso.php*

### Página de alta ( *alta.php* )

Esta página ha de ser un simple formulario en la que el usuario introduzca los datos para dar de alta una nueva cuenta indicando nº de cuenta, titular, saldo inicial e interés.

#### Validaciones:

El campo nº de cuenta admite 8 caracteres exactos.

El campo título admite máximo 50 caracteres.

Los campos saldo e interés deben ser valores numéricos válidos.

#### Comprobaciones:

Debe comprobarse que el nº de cuenta indicado no esté ya registrado. En caso de ser así debe mostrarse un mensaje de error manteniendo los valores introducidos en los campos.

### Página de listado de cuentas ( *listado.php* )

Esta página muestra un pequeño formulario y una tabla.

El formulario permite introducir el nombre de un titular para que se muestren únicamente sus cuentas bancarias al pulsar el botón "FILTRAR".

La tabla muestra el nº de cuenta, el titular, el saldo de todas las cuentas bancarias por defecto, salvo que se filtre por un titular. El campo correspondiente al nº de cuenta debe ser un hipervínculo que permita ir a la página de detalles ( *detalles.php* ) pasando como argumento el nº de cuenta.

### **Página de detalles ( *detalles.php* )**

Esta página muestra todos los datos de una determinada cuenta bancaria incluyendo nº de cuenta, titular, saldo, interés, su estado ( si está bloqueada o no ), y la fecha de alta.

La página debe constar además de un botón “*BLOQUEAR*” que permite marcar la cuenta bancaria como bloqueada. Esto impedirá que puedan realizarse operaciones sobre su saldo.

### **Página de movimientos ( *movimiento.php* )**

Esta página permite ingresar o retirar capital de una cuenta bancaria. Para ello debe incluirse un formulario que conste de los siguientes campos:

- Una lista desplegable de nº de cuentas de todas las cuentas registradas.
- Una caja de texto para introducir la cantidad de capital
- Un área de texto para indicar el concepto del movimiento.
- Dos botones “*INGRESAR*” y “*RETIRAR*” que permiten ingresar o retirar la cantidad indicada de la cuenta bancaria.

#### Validaciones:

- El capital indicado debe ser una cifra numérica válida positiva.
- Debe haberse seleccionado un nº de cuenta en la lista.
- Debe indicarse algo en el concepto.

#### Comprobaciones:

- No puede retirarse ni ingresarse capital en una cuenta si está bloqueada. De ser así debe indicarse mediante un mensaje de error “*CUENTA BLOQUEADA*”.
- No puede retirarse más capital que el saldo presente en la cuenta. De intentarlo debe mostrarse un mensaje de error “*SALDO INSUFICIENTE*”.

Si todo es correcto debe crearse el registro correspondiente al movimiento en la tabla *MOVIMIENTOS* y actualizar el saldo del registro de la cuenta en la tabla *CUENTAS*.

### **Página de traspasos ( *traspaso.php* )**

Esta página permite realizar un traspaso de capital entre dos cuentas de la misma entidad bancaria. Para ello la página consta de un formulario con los siguientes elementos.

- Una lista desplegable “*cuenta de origen*” con los nº de cuenta de todas las cuentas registradas en la base de datos.
- Una lista desplegable “*cuenta de destino*” con los nº de cuenta de todas las cuentas registradas en la base de datos.
- Una caja de texto para introducir la cantidad de capital a traspasar de la cuenta de origen a la cuenta de destino.



### Validaciones:

- Debe haberse seleccionado tanto un nº de cuenta de origen como uno de destino.
- El capital indicado debe ser una cifra numérica válida positiva.
- Los nº de cuenta destino y origen seleccionados no pueden ser el mismo.

### Comprobaciones:

- El traspaso no puede realizarse si el capital para el traspaso supera el saldo de la cuenta de origen. En tal caso debe indicarse un mensaje de error *"SALDO INSUFICIENTE EN CUENTA DE ORIGEN"*.
- El traspaso no puede realizarse si alguna de las cuentas indicadas ( origen / destino ) está bloqueada. En tal caso debe indicarse un mensaje de error *"LA CUENTA XXXXX ESTÁ BLOQUEADA"*.

Si todo es correcto deben generarse dos registros de movimiento en la tabla *MOVIMIENTOS* correspondiente a ambas cuentas.

- En el movimiento de la cuenta de origen debe indicarse el saldo traspasado (en negativo) y el concepto *"TRASPASO A "...* seguido del nº de cuenta de la cuenta de destino.
- En el movimiento de la cuenta de destino debe indicarse el saldo traspasado (en positivo) y el concepto *"TRASPASO DESDE "...* seguido del nº de cuenta de origen.
- En ambos casos, debe actualizarse igualmente el saldo de ambas cuentas en la tabla *CUENTAS*.

## Autenticación de usuarios

La autenticación de usuarios es un mecanismo que se emplea para identificar a los usuarios que tienen permiso para acceder una aplicación web o parte de ella. Para ello los usuarios deben indicar unas credenciales que permite autenticar su identidad y permitirle el acceso si tiene autorización.

Un ejemplo básico de autenticación sería emplear un formulario que solicita al usuario su nombre y contraseña para enviarlo a un script de PHP que comprueba la identidad del usuario y muestra la página de inicio o una página de error.

**Ejemplo:** Los siguientes códigos muestran una página de inicio (*access.html*) provista de un formulario en el que se solicita al usuario su nombre y contraseña. Los datos son enviados al script de control de acceso ( *access.php* ) que los comprueba y envía como respuesta al usuario la página de inicio ( *home.html* ) si son correctos, o una página de error ( *error.html* ) en caso contrario.

<b>access.html</b>
<pre> &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;meta charset="ISO-8859-1"&gt; &lt;title&gt;Insert title here&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;CONTROL DE ACCESO&lt;/h1&gt; &lt;form action="access.php" method="post"&gt;   &lt;p&gt;USUARIO&lt;input type="text" name="usuario"&gt;&lt;/p&gt;   &lt;p&gt;CONTRASEÑA&lt;input type="password" name="clave"&gt;&lt;/p&gt;   &lt;input type="submit" value="Comprobar"&gt; &lt;/form&gt; &lt;/body&gt; &lt;/html&gt; </pre>
<b>access.php</b>
<pre> &lt;?php // Comprobacion de valores enviados if ( count(\$_POST) &gt; 0 ) {     // Obtencion de valores     \$usuario = filter_input(INPUT_POST, "usuario");     \$clave = filter_input(INPUT_POST, "clave");     // Comprobacion de valores     if ( \$usuario == "usuario" &amp;&amp; \$clave == "clave" ) {         // Todo correcto --&gt; carga de la página de inicio         header('Location: '. 'home.html');     } else {         // Error --&gt; Parámetros de usuario incorrectos.         header('Location: '. 'error.html');     } } else {     // Error --&gt; Parámetros de usuario inexistentes.     header('Location: '. 'error.html'); } </pre>

Este mecanismo es muy sencillo y no demasiado seguro. Un usuario malintencionado podría acceder directamente a la página de inicio (*home.html*) a partir de su URL una vez que la conozca sin autenticarse.

Para proteger el acceso a múltiples páginas de modo que sólo un usuario autenticado pueda acceder a ellas aun haciendo referencia a sus URLs directamente en el navegador es necesario un mecanismo más avanzado que pueda:

- Almacenar el estado del usuario tras pasar por la página de autenticación de modo que pueda comprobarse en cada página a la que acceda.
- Recuperar y comprobar en cada página el estado del usuario para verificar si está autenticado y quién es. De este modo si un usuario malintencionado intenta acceder a una página sin autenticarse primero; es rechazado.

Para ello es necesario emplear el **estado de sesión**.

### Estado de sesión

El protocolo HTTP es un protocolo sin estado que no almacena ni mantiene ninguna información entre peticiones. Esto supone que cada petición al servidor se trata de manera independiente del resto siendo imposible por defecto identificar el usuario del que proceden.

El control de sesión permite identificar las peticiones de cada usuario asociándole un **identificador de sesión**, y un conjunto de datos. Estos datos constituyen el **estado de la sesión** y son accesibles por el código PHP de las página solicitadas por dicho usuario exclusivamente.

El identificador de sesión (*también llamada SID*) es una cadena de 32 caracteres que se almacena en una cookie especial denominada *PHPSESSID* en el navegador del usuario, y se envía al servidor web con cada petición. El servidor identifica entonces el SID de cada petición y obtiene el estado de sesión de ese usuario.

### Cookies ( ¿alguien tiene hambre? )

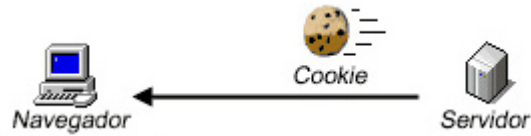
Las cookies son pequeños archivos que pueden almacenar cualquier conjunto de valores codificados como una cadena de texto de longitud limitada. Estos ficheros se almacenan en el disco duro del ordenador del usuario y no consumen recursos en el servidor web. Sin embargo, tienen ciertas limitaciones:

- El almacenamiento de cookies pueden estar deshabilitado.
- Las cookies pueden ser eliminadas por el usuario en cualquier momento.
- Los datos de las cookies pueden ser alterados por usuarios malvados.

Por todo ello, no son fiables ni seguras, y no se recomienda su uso para almacenar datos sensibles como cuentas de usuario, contraseñas..., etc.

## Programación Web en PHP

Las cookies son enviadas al navegador del usuario en la cabecera del mensaje de respuesta HTTP en el que se envían las páginas desde el servidor Web. Por su lado; el navegador almacena las cookies asociadas al dominio de la página de donde proceden.



Cuando el usuario solicita una página perteneciente a un dominio, el navegador envía al servidor web las cookies asociadas a dicho dominio como parte de la petición HTTP.



Esto implica que almacenar muchas cookies en el cliente provocará un mayor tiempo de carga en cada petición al servidor empobreciendo la navegación entre páginas de la aplicación web.

### Envío de cookies al cliente con PHP

Una cookie almacena los datos asociados a un valor clave. Esta clave es necesaria para identificar los valores a recuperar posteriormente. PHP posee la función **setcookie** que permite tanto crear como eliminar cookies en el navegador del usuario.

```
bool setcookie ( string $name
                [, string $value
                [, int $expire = 0
                [, string $path
                [, string $domain
                [, bool $secure = false
                [, bool $httponly = false ]]]]] )
```

Los parámetros más importantes son los siguientes:

- **\$name** → Indica la clave del valor que se añade a la cookie
- **\$valor** → Indica el valor asociado a la clave que se almacena en la cookie. Si se omite el dato se elimina la cookie.
- **\$expire** → El tiempo en el que expira la cookie. Es una fecha Unix por tanto está en número de segundos a partir de la presente época. En otras palabras, probablemente utilizará la función `time()` más el número de segundos que quiere que dure la cookie. También podría utilizar la función `mktime()`. `time()+60*60*24*30` configurará la cookie para expirar en 30 días. Si se pone 0, o se omite, la cookie expirará al final de la sesión (al cerrarse el navegador).

El resto de los parámetros pueden consultarse en la web oficial:

<http://php.net/manual/es/function.setcookie.php>

### Ejemplos

```
// Inserta en la cookie el valor "dato" con la clave "TestCookie"
setcookie("TestCookie", "dato");

// Inserta en la cookie el valor "dato" con la clave "TestCookie". La
// cookie caduca en 1 hora, tras la cual expira y es eliminada.
setcookie("TestCookie", $value, time()+3600);
```

La eliminación de las cookies puede llevarse a cabo invocando la función **setcookie()** indicando únicamente el identificador de la cookie. El mismo efecto se consigue modificando la cookie con un tiempo de expiración negativo, lo que fuerza al navegador a eliminarla por haber expirado:

```
// Eliminación de la cookie con identificador: "TestCookie".
Setcookie("TestCookie");

// No se añade ningún valor a la cookie. Su tiempo de expiración
// se pone en negativo provocando su eliminación por el navegador.
setcookie ("TestCookie", "", time() - 3600);
```

La actualización de cookies consiste en modificar su valor enviando una cookie con el mismo identificador. Esto eliminará el valor anterior a favor del nuevo.

Las cookies se envían al navegador en la cabecera de la respuesta HTTP, por lo que deben declararse siempre al principio de la página antes del envío de código HTML.

### Obtención de cookies desde el cliente con PHP

Cuando el navegador envía una petición de una página a un servidor Web, envía al mismo tiempo la información de las cookies asociadas a la URL en la propia petición. De este modo, el valor de las cookies puede ser recuperado desde el código PHP.

PHP define la variable superglobal **\$\_COOKIE** que almacena una matriz asociativa con los identificadores y valores de todas las cookies enviadas por el navegador. Su uso es idéntico al de otras matrices como **\$\_POST**, **\$\_GET**, **\$\_REQUEST**.

**Ejemplo:** El siguiente código recupera el valor de la cookie con el identificador "id":

```
// Comprueba si existe la cookie con el identificador "id".
if (isset( $_COOKIE["id"] ))
{
    // Recupera el valor de la cookie con el identificador "id".
    $valor = $_COOKIE["id"];
}
```

Es importante antes de recuperar el valor de una cookie comprobar primero si existe utilizando por ejemplo la función **isset()**, ya que puede ser la primera vez que el usuario acceda a la página, o bien haber expirado las cookies almacenadas anteriormente.

## Manejo del estado de sesión.

Como ya se ha comentado anteriormente, PHP mantiene el estado de sesión asociando a cada usuario un *identificador de sesión* que se almacena mediante una cookie en su navegador. Esta cookie es gestionada por PHP. Los datos del estado de sesión se almacenan sin embargo en el servidor web permaneciendo seguros.

Antes de poder guardar y recuperar datos del estado de sesión es necesario indicar a PHP que se inicie la sesión. Esto se realiza llamando a la función ***session\_start()***. Esta función realiza las siguientes operaciones:

- Si no hay una sesión asociada al usuario (es la primera página que solicita), se le envía la cookie *PHPSESSID* en la respuesta.
- Si ya existe una sesión, la función recupera el identificador de sesión y asocia los datos del estado de sesión de ese usuario en la matriz superglobal *\$\_SESSION*.

La función ***session\_start()*** debe invocarse en cada página que haga uso del estado de sesión siempre al principio del código ya que añade la cookie *PHPSESSID* a la cabecera del mensaje de respuesta HTTP.

### Registro y recuperación de valores en el estado de sesión

La inserción y recuperación de datos en el estado de sesión se lleva acabo empleando la variable superglobal ***\$\_SESSION***. Esta variable se corresponde con una matriz asociativa que permite insertar y obtener cualquier valor asociado a una clave.

**Ejemplo:** El siguiente código inicia el estado de sesión y almacena un valor asociado a la clave "id":

```
// Inicio del estado de sesión
session_start();
// Inserción de datos en el estado de sesión
$_SESSION['id'] = 'alumno';
```

**Ejemplo:** El siguiente código muestra la recuperación del valor almacenado previamente del estado de sesión con el identificador "id":

```
// Inicio de la sesion
session_start();
// Existe el valor en el estado de sesion
if ( isset($_SESSION['id'])) {
    // Recuperacion del valor
    $valor = $_SESSION['id'];
}
```

Dado que ambos códigos se supone que irían situados en scripts diferentes, ambos incluyen la llamada a la función ***session\_start()*** para poder manejar la matriz superglobal ***\$\_SESSION***.

### Eliminación del estado de sesión

Los valores almacenados en el estado de sesión ocupan memoria en el servidor Web. Por ello es conveniente eliminar valor cuando no sean necesarios empleando la función ***unset()***:

```
// Eliminacion de un valor del estado de sesion
unset($_SESSION['id']);
```

Si se desean eliminar todos los valores almacenados en el estado de sesión del usuario puede emplearse el siguiente código:

```
// Eliminacion de todos los valores del estado de sesion
$_SESSION = array();
```

El propio estado de sesión puede ser eliminado cuando el usuario cierra sesión y se desea que las siguientes solicitudes se asocien a uno nuevo usuario. Para ello debe llamarse al método ***session\_destroy()***:

```
// Eliminación del estado de sesion
session_destroy();
```

**Ejemplo:** Esta página muestra un formulario en la que el usuario puede introducir un valor que es almacenado en su estado de sesión con el identificador 'id' para ser recuperado en otras páginas.

#### ***session\_reg.php***

```
<?php
    // Sesión iniciada
    session_start();
    // Valor indicado?
    if ( isset($_POST['valor'])) {
        // Registro de valor en estado de sesión
        $_SESSION['id'] = filter_input(INPUT_POST, 'valor');
    }
?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form method="post">
            <input type="text" name="valor">
            <input type="submit">
        </form>
        <a href="session_show.php">MOSTRAR SESION</a>
        <a href="session_unreg.php">ELIMINAR SESION</a>
    </body>
</html>
```

**Ejemplo:** La siguiente página muestra el valor almacenado en el estado de sesión del usuario con el identificador 'id'. Si no existe, muestra el mensaje "NO HAY DATOS EN SESION".

### **session\_show.php**

```
<?php
    // Sesion iniciada
    session_start();
    // Valor en estado de sesion?
    if ( isset($_SESSION['id']))
    {
        // Recuperación del valor
        $valor = $_SESSION['id'];
    } else {
        $valor = "NO HAY DATOS EN LA SESION";
    }
?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <h1><?= htmlspecialchars($valor, ENT_NOQUOTES, 'utf-8'); ?></h1>
        <a href="session_reg.php">REGISTRAR SESION</a>
        <a href="session_unreg.php">ELIMINAR SESION</a>
    </body>
</html>
```

**Ejemplo:** La siguiente página destruye el estado de sesión del usuario eliminando todos los valores que hubiera almacenados en él.

### **session\_unreg.php**

```
<?php
    // Sesion iniciada
    session_start();
    // Sesion eliminada
    session_destroy();
?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <h1>SESION ELIMINADA</h1>
        <a href="session_reg.php">REGISTRAR SESION</a>
        <a href="session_show.php">MOSTRAR SESION</a>
    </body>
</html>
```



## Autentificación empleando estado de sesión

El empleo del estado de sesión permite almacenar unos datos asociados a cada usuario que pueden recuperarse en cada página que solicite sin necesidad de enviarlos de una página a la siguiente. Esto permite mejorar el mecanismo de autentificación visto anteriormente de modo que puedan protegerse múltiples páginas de accesos no autorizados.

El formulario inicial de autentificación no sufre ningún cambio.

```
access.html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>CONTROL DE ACCESO</h1>
<form action="access.php" method="post">
  <p>USUARIO<input type="text" name="usuario"></p>
  <p>CONTRASEÑA<input type="password" name="clave"></p>
  <input type="submit" value="Comprobar">
</form>
</body>
</html>
```

Los datos con las credenciales del usuario son enviados al script de acceso (*access.php*). Este debe ocuparse de las siguientes tareas:

- Obtener los datos del formulario
- Comprobar si se corresponden con los de un usuario con permiso de acceso,
- Registrar en su estado de sesión que está autentificado.

Habitualmente, las credenciales de los usuarios con permisos de acceso suelen almacenarse en una base de datos. Para ello, supongamos que en la base de datos 'bancos' disponemos de una tabla **USUARIOS** que almacena el nombre de usuario y contraseña de todos los usuarios con permiso de acceso a la aplicación web:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
1	USUARIO	varchar(10)			No	Ninguna	
2	CLAVE	varchar(10)			No	Ninguna	
3	ADMINISTRADOR	tinyint(1)			No	0	

La columna **ADMINISTRADOR** permite indicar si el usuario tiene permisos de administración especiales o no sobre la aplicación web. Esto permite crear diferentes niveles de seguridad, por ejemplo; un administrador podría dar de alta y baja cuentas bancarias mientras que un usuario ordinario no.

Para acceder a la base de datos emplearemos la clase DB (*db.class.php*) vista anteriormente que nos sirve de *wrapper* para simplificar el código:

```
access.php
<?php
require 'db.class.php';
session_start();
// Comprobacion de valores enviados
if ( count($_POST) > 0 ) {
    // Obtencion de valores
    $usuario = filter_input(INPUT_POST, "usuario");
    $clave = filter_input(INPUT_POST, "clave");
    // Comprobacion de valores
    // Acceso a la base de datos
    $resultado = DB::run("SELECT USUARIO, CLAVE, ADMINISTRADOR FROM USUARIOS WHERE
    USUARIO = :usuario AND CLAVE = :clave", [':usuario' => $usuario, ':clave' => $clave]);
    // Obtencion de datos del usuario con las credenciales indicadas
    if ( $registro = $resultado->fetch(PDO::FETCH_ASSOC)) {
        // Existe el usuario. Sus datos se almacenan en el estado de sesion
        $_SESSION['usuario'] = $registro;
        header('Location: '. 'home.php');
    } else {
        // No existe el usuario. Usuario desconocido
        header('Location: '. 'error.html');
    }
} else {
    // Error --> Parámetros de usuario inexistentes.
    header('Location: '. 'error.html');
}
```

Para saber si existe el usuario en la tabla **USUARIOS** se hace una consulta buscando el registro con el nombre de usuario y contraseña dados. Si la consulta devuelve un registro, es que el usuario existe y se guardan sus datos en el estado de sesión para denotar que se ha autenticado y se le redirecciona a la página de inicio (*home.php*). Si la consulta no devuelve ningún registro, es que no hay ningún usuario registrado con los datos indicados y se redirecciona el usuario a la página de error.

A continuación, todas las página que requieran que el usuario esté autenticado para acceder a ellas deben comprobar el estado de sesión para ver si efectivamente el usuario se ha autenticado correctamente.

```
session.php
<?php
// Inicialización del estado de sesion
session_start();
// Existen datos del usuario en el estado de sesion?
if ( isset($_SESSION['usuario'])) {
    // Existen: Se almacenen en la variable $usuario
    $usuario = $_SESSION['usuario'];
} else {
    // No existen: Redireccion a la página de error.
    header('Location: '. 'error.html');
}
```

Este código debería incluirse al principio de la página de inicio (*home.html*), por lo que debe convertirse a una página PHP (*home.php*), igual que el resto de páginas que queramos proteger.

Para no repetir el código anterior en todas las páginas, lo mejor es añadirlo a un script (*session.php*) e incluirlo en las páginas mediante un simple **require**:

Con ello, el código de la página de inicio queda del siguiente modo:

<b>home.php</b>
<pre>&lt;?php     require 'session.php'; ?&gt;  &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;meta charset="ISO-8859-1"&gt; &lt;title&gt;Insert title here&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;BIENVENIDO &lt;?= \$usuario['USUARIO'] ?&gt;&lt;/h1&gt; &lt;p&gt;&lt;?= (\$usuario['ADMINISTRADOR'])?"ADMINISTRADOR":"USUARIO"; ?&gt;&lt;/p&gt; &lt;a href="Logout.php"&gt;CERRAR SESION&lt;/a&gt; &lt;/body&gt; &lt;/html&gt;</pre>

La página incluye el script de control de autenticación (*session.php*), y muestra los datos del usuario autenticado empleando la variable *\$usuario*. Si el script de autenticación detecta que el usuario está autenticado almacena en *\$usuario* sus datos. En caso contrario, provoca un redireccionamiento a la página de error. De este modo, si un usuario malintencionado intente acceder a la página de inicio (home.php) sin autenticarse antes, sólo verá la página de error.

Finalmente, la página de cierre de sesión (*logout.php*) permite al usuario cerrar sesión para permitir que otro usuario se autentique en la aplicación en el mismo ordenador.

<b>logout.php</b>
<pre>&lt;?php     require 'session.php';     // Eliminacion de la sesion     session_destroy(); ?&gt;  &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;meta charset="ISO-8859-1"&gt; &lt;title&gt;Insert title here&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;HAS CERRADO SESION &lt;?= \$usuario['USUARIO'] ?&gt;&lt;/h1&gt; &lt;a href="access.html"&gt;VOLVER A FORMULARIO DE ACCESO&lt;/a&gt; &lt;/body&gt; &lt;/html&gt;</pre>

## Configuración del estado de sesión

En el archivo de configuración de PHP (php.ini) existen una serie de parámetros que permiten configurar ciertos aspectos del manejo del estado de sesión que pueden resultar importantes:

- ***session.auto\_start*** → Este parámetro indica si automáticamente todas las páginas inician el estado de sesión como si ejecutasen ***session\_start()***. Por defecto, está desactivado.
- ***session.cache\_expire*** → Este parámetro indica el *tiempo de expiración de la sesión*. Este es el tiempo máximo que se mantiene el estado de sesión de un usuario en el servidor a partir de su última petición. Transcurrido el tiempo, el usuario debería volver a autenticarse. El tiempo predeterminado son 180 segundos.
- ***session.cookie\_lifetime*** → Este parámetro indica el tiempo de expiración de la cookie con el identificador de sesión en el navegador del usuario. El valor por defecto es 0, de modo que se elimina al cerrar el navegador. Esto provocaría la pérdida del identificador de sesión y el usuario debería volver a autenticarse para acceder a la aplicación web.
- ***session.use\_cookies*** → Este parámetro indica si el identificador de sesión se almacena empleando una cookie. El valor predeterminado es 1 habilitando el empleo de cookies.

### Mantenimiento del identificador de sesión sin cookies

Las cookies pueden no estar habilitadas en el navegador del usuario. En este caso, no pueden emplearse para almacenar el identificador de sesión.

En estas circunstancias el único modo de almacenar el identificador de sesión que permite identificar las peticiones de cada usuario es concatenar el propio identificador a la URL de sus peticiones. Para ello podemos emplear las siguientes sentencias:

```
ini_set('session.use_cookies', 0);  
ini_set('session.use_only_cookies', 0);
```

Estas sentencias desactivan el uso de cookies para mantener el identificador de sesión por parte de PHP. Ahora, debemos modificar los enlaces y redireccionamientos añadiendo a la URL el identificador de sesión:

Modificación en *access.php* para redireccionar a la página de inicio (home.php) pasando identificador de sesión en la URL:

```
header('Location: '.'home.php?'.htmlspecialchars($SID));
```

Modificación en *home.php* en enlace a página "*logout.php*" pasando identificador de sesión en la URL:

```
<a href="logout.php?<?php echo htmlspecialchars($SID);?>">CERRAR SESION</a>
```

Las URL generadas tienen el siguiente aspecto:

***http://localhost/autenticacion/home.php?PHPSESSID=15bht7lcttohdm888rtpnc8gf3***

**(\*) El mantenimiento del identificador de sesión mediante URLs es posible, pero mucho más inseguro, ya que un usuario malintencionado podría suplantar a uno autenticado copiando el identificador de sesión de su URL mientras ésta no haya expirado. NO DEBE EMPLEARSE**

## ***Practica***

Se pide modificar la aplicación web de gestión bancaria creada anteriormente añadiendo un formulario inicial que permite autenticar a los usuarios, y sólo permita acceder a la aplicación web a aquellos registrados en la tabla *USUARIOS* de la base de datos '*bancos*' implementada anteriormente.

Deben tenerse en cuenta los siguientes requisitos:

- Todas las páginas de la aplicación web deben protegerse de modo que ningún usuario malintencionado pueda acceder a ellas sin haberse autenticado previamente. En tal caso debe redirigirse al usuario al formulario de autenticación.
- Todas las páginas deben añadir una cabecera indicando el nombre del usuario autenticado.
- Debe añadirse en todas las páginas un enlace "CERRAR SESION" que permita al usuario salir de la aplicación retornando al formulario de autenticación tras destruirse su estado de sesión.
- El alta de cuentas bancarias sólo está permitida a usuarios administradores, por lo que sólo debe ser accesible a aquellos usuarios con permiso de administrador ( deben tener el valor '1' en el campo *ADMINISTRADOR* en la tabla *USUARIOS* ).