PHP. El nuevo PHP. Conceptos avanzados
Autores: Eslava Muñoz, J.
Bubok, 2013, págs. 9-25
ISBN: 9788468644332

PHP

El nuevo PHP. Conceptos avanzados.

Vicente Javier Eslava Muñoz

El nuevo PHP.
Conceptos avanzados.

Por Vicente Javier Eslava Muñoz

1 – Acceso a base de datos.

Una de las principales razones de la popularidad de PHP como lenguaje de creación de scripts para Web es su amplio soporte a diferentes bases de datos. Este soporte facilita que los desarrolladores creen sitios sustentados en bases de datos y que se hagan nuevos prototipos de aplicaciones Web de manera rápida y eficiente, sin demasiada complejidad.

PHP soporta más de quince diferentes motores de bases de datos, incluidos Microsoft SQL Server, IBM DB2, PostgreSQL, MySQL y Oracle. Hasta PHP 5, este soporte se proporcionaba mediante extensiones nativas de las bases de datos, cada una con sus propias características y funciones; sin embargo, esto dificultaba a los programadores el cambio de una base a otra. PHP 5 rectificó esta situación introduciendo una API común para el acceso a base de datos: las extensiones de objetos de datos de PHP (PDO, *PHP Data Objects*), que proporcionan una interfaz unificada para trabajar con bases de datos y ayudan a que los desarrolladores manipulen diferentes bases de datos de manera consistente.

Las extensiones PDO han sido mejoradas, con soporte para más motores de bases de datos y mejoras considerables en la seguridad y el desempeño. Para fines de compatibilidad con versiones anteriores, se sigue dando soporte a las extensiones de bases de datos nativas. En ocasiones se tendrá que escoger entre una extensión nativa (que puede ser más veloz u ofrecer más características) y una PDO (que ofrece portabilidad y consistencia para diferentes motores de bases de datos. A continuación se presentan ambas opciones en detalle.

Aclaremos ciertos conceptos previos:

*¿Qué es una **API**?*

Una Interfaz de Programación de Aplicaciones (o API de sus siglas en inglés), define las clases, métodos, funciones y variables que la aplicación necisita llamar para realizar una tarea. En el caso de aplicaciones de PHP que necesiten comunicarse con bases de datos, las APIs necesarias normalmente son expuestas mediante extensiones de PHP.

Las APIs pueden ser procedimentales y orientadas a objetos. Con una API procedimental se llaman a funciones para realizar tareas, con una API orientada a objetos se instancian clases y luego se llamana métodos sobre los objetos resultantes. De las dos, la última normalemte es la interfaz preferida, ya que es más moderna y conduce a un código mejor organizado.

Al escribir aplicaciones de PHP que necesitan conectarse a un servidor de MySQL, existen varias opciones de APIs disponibles. Este documento trata sobre lo que está disponible y cómo elegir la mejor solución para la aplicación.

*¿Qué es un **Conector**?*

En la documentación de MySQL, el término *conector* se refiere a una pieza de software que permite a las aplicaciones conectarse con el servidor de bases de datos MySQL. MySQL proporciona conectores para muchos lenguajes, incluido PHP.

Si una aplicación de PHP necesita comunicarse con un servidor de bases de datos se necesitará escribir código de PHP para llevar a cabo actividades como conectar

al servidor de la base de datos, consultar a la base de datos y otras funciones relacionas con la base de datos. Se requiere software para proporcionar la API que la aplicacion de PHP usraá, y también manejar la comunicación entre la aplicación y el servidor de la base de datos, posiblemente usanto otras bibliotecas intermediarias cuando sea necesario. Este software se conoce generalmente como conector, ya que permite a la aplicación *conectarse* al servidor de la base de datos.

*¿Qué es un **Controlador**?*

Un controlador (o driver) es una pieza de software diseñada para la comunicación con un tipo específico de servidor de bases de datos. El controlador también llama a una biblioteca, como la Biblioteca Cliente de MySQL o el Controlador Nativo de MySQL. Estas bibliotecas implementan el protocolo de bajo nivel usado para comunicarse con el servidor de bases de datos MySQL.

Mediante un ejemplo, la capa de abstracción de bases de datos Objetos de Datos de PHP (PDO) pueden usar uno de los varios controladores específicos de bases de datos. Uno de los controladores disponibles es el controlador MySQL de PDO, que le permite funcionar junto con el servidor MySQL.

A veces las personas usan los términos conector y controlador intercambiablemente, y esto puede ser confuso. El la documentación relacionada con MySQL, el término "controlador" está reservado para el software que proporciona la parte estpecífica de bases de datos de un paquete conector.

*¿Qué es una **Extensión**?*

En la documentación de PHP se encontrará con otro término - *extensión*. El código de PHP consiste en un núcleo, con extensiones opcionales para la funcionalidad del núcleo. Las extensiones relacionadas con MySQL de PHP, como la extensión *mysqli*, y la extensión *mysql*, están implementadas usando el framework de extensiones de PHP.

Una extensión típicamente expone una API al programador de PHP, para poder usar sus facilidades programáticamente. Sin embargo, algunas extensiones que usan el framework de extensiones de PHP no exponen una API al programador de PHP.

La extensión del controlador MySQL de PDO, por ejemplo, no expone una API al programador de PHP, pero proporciona una interfaz para la capa de PDO superior.

Los términos API y extensión no deberían tomarse como si fuesen la misma cosa, una extensión puede no exponer necesariamente una API al programador.

ATENCIÓN: Se recomienda al usuario si no está acostumbrado a trabajar con bases de datos y a utilizar el lenguaje SQL que revise los Anexos I y II.

1.1 – Extensión nativa MySQLi.

De los diferentes motores de base de datos soportados por PHP, el más popular es MySQL. No resulta difícil entender el porqué: tanto PHP como MySQL son proyectos

de código libre, y al utilizarlos juntos, los desarrolladores obtienen beneficios de los grandes ahorros en costos de licencias en comparación con las opciones comerciales.

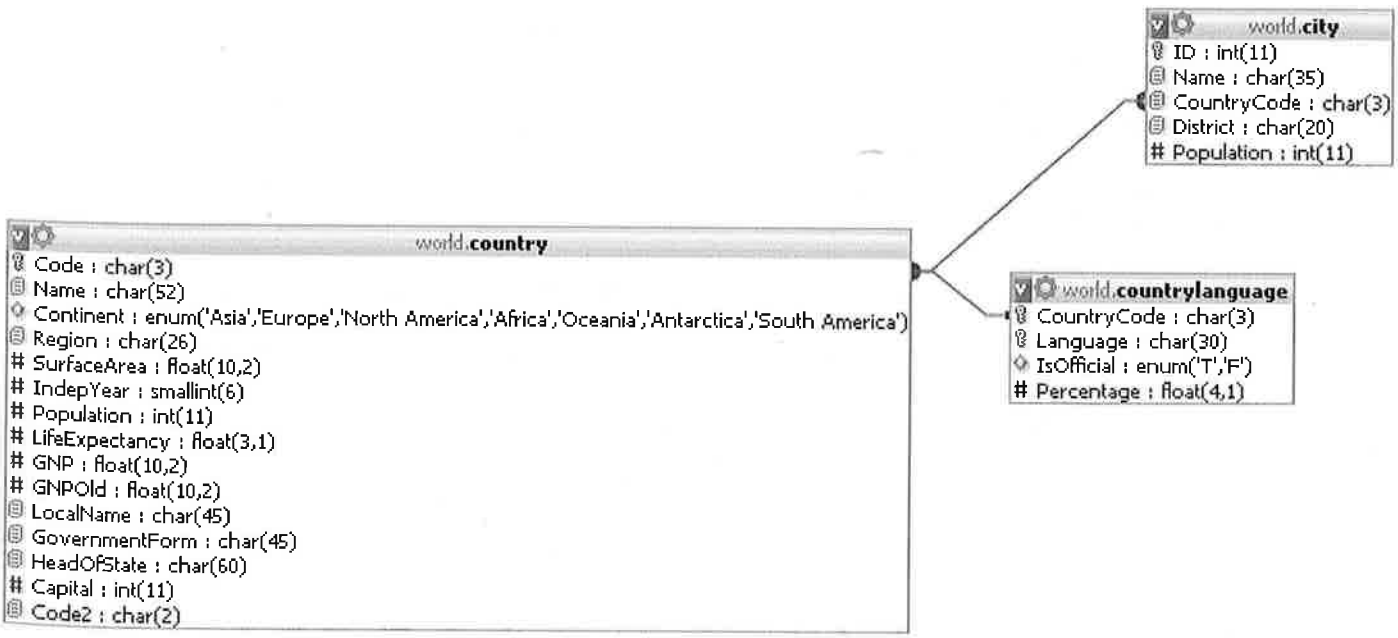
MySQLi es una extensión nativa para trabajar con las bases de datos MySQL de forma más eficiente que con PDO (por el contrario el código será menos exportable a otras bases de datos). MySQLi es la evolución de la antigua extensión MySQL (MySQLi = MySQL improved) que no debe ya utilizarse en proyectos por quedar obsoleta.

La extensión *mysqli* contiene numerosos beneficios, siendo estas las mejoras principales respecto a la extensión *mysql*:

- Interfaz orientada a objetos
- Soporte para Declaraciones Preparadas
- Soporte para Múltiples Declaraciones
- Soporte para Transacciones
- Mejoradas las opciones de depuración
- Soporte para servidor empotrado

Se puede encontrar más información sobre MySQL en » http://www.mysql.com/

ATENCIÓN: Los ejemplos siguientes se basan sobre la base de datos **world** que se puede encontrar en http://downloads.mysql.com/docs/world.sql.gz



Conectando con la base de datos.

Una aplicación Web con PHP que utiliza por detrás una BD debe realizar básicamente los siguientes pasos (pasos genéricos – Siempre ocurren):

- Establecer una conexión con la BD
  - Crear una sentencia SQL
    - Ejecutar la sentencia SQL
      - Procesar el resultado
- Terminar la conexión

Las conexiones debemos realizarlas en cada script en el que utilicemos o debamos hacer algún tipo de consulta.

Con el fin de establecer comunicación con el servidor que contiene la base de datos MySQL, primero necesitas abrir una conexión con el mismo. Toda la comunicación entre PHP y el servidor de base de datos se realiza a través de esta conexión.

Para inicializar esta conexión, inicializa un objeto de la clase MySQLi y pasa cuatro argumentos al constructor del objeto: el nombre del servidor anfitrión de MySQL al que intentas conectarte, un nombre y una contraeña válidos para obtener el acceso necesario, y el nombre de la base de datos que quieres utilizar.

MySQLi abre una conexión al Servidor MySQL Server sobre el que se ejecuta, devuelve un objeto que representa la conexión al servidor MySQL y tiene los siguientes parámetros:

host → Puede ser o un nombre de host o una dirección IP. Pasando el valor **NULL** o la cadena "localhost" a este parámetro, se asumirá el host local. Cuando es posible, se usarán tuberías en lugar del protocolo TCP/IP.

username → El nombre de usuario de MySQL.

passwd → Si no se proporciona o es **NULL**, el servidor MySQL intentará autentificar el usuario solo con aquellos registros de usuarios que no tienen contraseña. Esto permite que un nombre de usuario ser usado con diferentes permisos (dependiendo de si se proporciona una contraseña o no).

dbname → Si se proporcioina, especificará la base de datos prederminada a usar cuando se realizan consultas.

port → Especifica el número al que intentar conectar al servidor de MySQL.

socket → Especifica el socket o la tubería con nombre que debería usarse.

Nota: Especificar el parámetro *socket* no determinará explicitamente el tipo de conexión a utilizar cuando se conecte al servidor MySQL. El modo de realizar la conexión a la base de datos MySQL es determinado por el parámetro *host*.

Podemos utilizar la clase MySQLi para conectar por medio de dos estilos diferentes: por procedimientos o orientado a objetos.

Estilo orientado a objetos

Sintaxis:

```
mysqli::__construct() (
        [ string $host = ini_get("mysqli.default_host")
        [, string $username = ini_get("mysqli.default_user")
        [, string $passwd = ini_get("mysqli.default_pw")
        [, string $dbname = ""
        [, int $port = ini_get("mysqli.default_port")
        [, string $socket = ini_get("mysqli.default_socket") ]]]]]
)
```

Ejemplo:

```
<?php
$mysqli = new mysqli('localhost', 'mi_usuario', 'mi_contraseña', 'mi_bd');
if ($mysqli->connect_error) {
    die('Error de Conexión (' . $mysqli->connect_errno . ') ' . $mysqli->connect_error);
}
echo 'Éxito... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

Ejemplo extendiendo la clase mysqli:

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $usuario, $contraseña, $bd) {
        parent::__construct($host, $usuario, $contraseña, $bd);
        if (mysqli_connect_error()) {
            die('Error de Conexión (' . mysqli_connect_errno() . ') ' . mysqli_connect_error());
        }
    }
}
$bd = new foo_mysqli('localhost', 'mi_usuario', 'mi_contraseña', '');
echo 'Éxito... ' . $bd->host_info . "\n";
$bd->close();
?>
```

Estilo por procedimientos

Sintaxis:

```
mysqli mysqli_connect (
        [ string $host = ini_get("mysqli.default_host")
        [, string $username = ini_get("mysqli.default_user")
        [, string $passwd = ini_get("mysqli.default_pw")
        [, string $dbname = ""
        [, int $port = ini_get("mysqli.default_port")
        [, string $socket = ini_get("mysqli.default_socket") ]]]]]
)
```

Ejemplo:

```
<?php
$enlace = mysqli_connect('localhost', 'mi_usuario', 'mi_contraseña', 'mi_bd');
if (!$enlace) {
    die('Error de Conexión (' . mysqli_connect_errno() . ') ' . mysqli_connect_error());
}
echo 'Éxito... ' . mysqli_get_host_info($enlace) . "\n";
```

```
mysqli_close($enlace);
?>
```

Utilizamos **connect_error** para detectar si se ha producido un error en el establecimiento de la conexión. Si ese fuera el caso se aborta la página mediante la función **die** mostrando un mensaje "Error de Conexión…". En caso contrario se mostraría el mensaje "Éxito…" acompañado de la información de la conexión.

Por último cerramos la conexión llamando al método **close()** liberando los recursos reservados anteriormente.

El resultado de los ejemplos sería: Éxito... MySQL host info: localhost via TCP/IP

Seleccionar datos.

Para seleccionar datos de nuestra base de datos, una vez abierta la conexión, hemos de utilizar **query()**, la cual realiza una consulta Retorna **FALSE** si hay fallas. Si una consulta del tipo *SELECT, SHOW, DESCRIBE* o *EXPLAIN* es exitosa la funcion **mysqli_query()** retornara El objeto de la clase mysqli_result. Para otras consultas **mysqli_query()** retornara **TRUE** si tiene exito.

Dispone de los siguientes parámetros:

link → Sólo estilo por procediminetos: Un identificador de enlace devuelto por mysqli_connect() o mysqli_init()

query → La cadena de la consulta a la base de dato. La data dentro de la consulta a la BD debe ser escapada apropiadamente.

Resultmode → Ya sea la constante **MYSQLI_USE_RESULT** o **MYSQLI_STORE_RESULT** dependiendo del resultado deseado. Por defecto se usa la constante **MYSQLI_STORE_RESULT**. Si se usa **MYSQLI_USE_RESULT** todas la llamadas poesteriore retornaran con un error will return error *Commands out of sync* al menos que llamae la funcion call mysqli_free_result(). Con **MYSQLI_ASYNC** (disponible conh mysqlnd), es posible hacer consulta de manera asincrona. mysqli_poll() se utiliza para obtener los resultados dicha consulta

Estilo orientado a objetos

Sintaxis:

```
mysqli::query ( string $query [, int $resultmode = MYSQLI_STORE_RESULT ] )
```

Ejemplo:

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* comprobar conexión*/
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
/* Crear una table que no existe, no devuelve un conjunto de datos */
if ($mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
```

```
    printf("Table myCity successfully created.\n");
}
/* Las consultas select devuelven un conjunto de datos que podemos comprobar mediante
num_rows                                                                          */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);

    /* liberamos el conjunto de resultados */
    $result->close();
}
$mysqli->close();
?>
```

Estilo por procedimientos

Sintaxis:

```
mysqli_query ( mysqli $link , string $query [, int $resultmode = MYSQLI_STORE_RESULT ] ))
```

Ejemplo:

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Crear una table que no existe, no devuelve un conjunto de datos */
if (mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}

/* Las consultas select devuelven un conjunto de datos que podemos comprobar mediante
num_rows                                                                          */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));

    /* liberamos el conjunto de resultados */

    mysqli_free_result($result);
}

mysqli_close($link);
?>
```

Utilizaremos num_rows para mostrar la cantidad de filas que se han seleccionado. El resultado de los ejemplos sería:

```
Table myCity successfully created.

Select returned 10 rows.
```

Procesando los datos.

Cuando ejecutamos la query esta devuelve un objeto de tipo **MySQLi_Result.** Este método regresa cada registro de la colección de resultados como una matriz que contiene llaves indexadas tanto numéricamente como por cadenas de caracteres; esto ofrece a los desarrolladores la conveniencia de hacer referencia a campos individuales de cada registro ya sea por índice o por nombre de campo.

Vamos a estudiar un ejemplo en el que devolvemos los 10 primeros países y sus respectivos continentes. La query en SQL sobre la base de datos world es la siguiente:

```
SELECT Name, Continent FROM  country LIMIT 0 , 10
```

El código del ejemplo es:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user" "contra", "world");
if ($mysqli === false) {
        die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// itera sobre colección de resultados
// muestra cada registro y sus campos
$sql = "SELECT Name, Continent FROM  country LIMIT 0 , 10";

if ($result = $mysqli ->query($sql)) {
        if ($result->num_rows > 0) {
                while($row = $result->fetch_array()) {
                        echo $row["Name"] . " --> " . $row["Continent"] . "<br/>\n";
                }
        $result->close();
        } else {
                echo "No se encontró ningún registro que coincida con su búsqueda.";
        }
} else {
        echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}
// cierra conexión
$mysqli->close();
?>
```

$result es un objeto tipo MySQLi_Result cuya sintaxis es la siguiente:

```
mysqli_result implements Traversable {
/* Propiedades */
      int $current_field ;
      int $field_count;
      array $lengths;
      int $num_rows;
/* Métodos */
      bool data_seek ( int $offset )
      mixed fetch_all ([ int $resulttype = MYSQLI_NUM ] )
      mixed fetch_array ([ int $resulttype = MYSQLI_BOTH ] )
      array fetch_assoc ( void )
      object fetch_field_direct ( int $fieldnr )
      object fetch_field ( void )
      array fetch_fields ( void )
      object fetch_object ([ string $class_name [, array $params ]] )
      mixed fetch_row ( void )
```

```
    bool field_seek ( int $fieldnr )
    void free ( void )
}
```

Explicación de los métodos más importantes:

- mysqli_result::$current_field — Obtener posición del campo actual de un puntero a un resultado
- mysqli_result::data_seek — Ajustar el puntero de resultado a una fila arbitraria del resultado
- mysqli_result::fetch_all — Obtener todas las filas en un array asociativo, numérico, o en ambos
- mysqli_result::fetch_array — Obtiene una fila de resultados como un array asociativo, numérico, o ambos
- mysqli_result::fetch_assoc — Obtiene una fila de resultado como un array asociativo
- mysqli_result::fetch_field_direct — Obtener los metadatos de un único campo
- mysqli_result::fetch_field — Retorna el próximo campo del resultset
- mysqli_result::fetch_fields — Devuelve un array de objetos que representan los campos de un conjunto de resultados
- mysqli_result::fetch_object — Devuelve la fila actual de un conjunto de resultados como un objeto
- mysqli_result::fetch_row — Obtener una fila de resultados como un array enumerado
- mysqli_result::$field_count — Obtiene el número de campos de un resultado
- mysqli_result::field_seek — Establecer el puntero del resultado al índice del campo especificado
- mysqli_result::free — Libera la memoria asociada a un resultado
- mysqli_result::$lengths — Retorna los largos de las columnas de la fila actual en el resultset
- mysqli_result::$num_rows — Obtiene el número de filas de un resultado

Si el número de filas que devuelve la consulta es superior a 0 if ($result->num_rows > 0) { vamos a recorrer los resultados con un bucle while y asignando en la variable $row mediante el método **fetch_array()** de MySQLi_Result cada una de las filas. Por último mostramos las cadenas Name y Continent del array $row.

fetch_array retorna un array de strings que corresponde a la fila obtenida o NULL si no hay más filas en el resultset. Podemos acceder mediante el nombre de las columnas seleccionadas mediante el SELECT o directamente por índice en un array. Podemos sustituir fácilmente la línea

```
echo $row["Name"] . " --> " . $row["Continent"] . "<br/>\n";
```

por la ínea:

```
echo $row[0] . " --> " . $row[1] . "<br/>\n";
```

Una vez terminado de trabajar con los datos debemos cerrar la conexión con close(); El resultado de la ejecución del script anterior bajo la base de datos world es el que se muestra en la siguiente imagen.

Existe otro método para recuperar registros: como objetos, utilizando el método **fetch_object()**. Aquí, cada registro se representa como un objeto y los campos de un registro se representan como propiedades del objeto. Después, los campos individuales pueden ser accesados utilizando la notación estándar $objeto->propiedad. Presenta los siguientes parámetros:

result → Sólo estilo por procedimientos: Un conjunto de identificadores de resultados devuelto por mysqli_query(), mysqli_store_result() o mysqli_use_result().

class_name → El nombre de la clase a instanciar, establecer las propiedades y devolver. Si no se especifica se devuelve un objeto **stdClass**.

params → Un array opcional de parámetros para pasar al constructor de los objetos de *class_name*.

Podemos utilizar fetch_object() por medio de dos estilos diferentes: por procedimientos o orientado a objetos.

Estilo orientado a objetos

Sintaxis:

```
object mysqli_result::fetch_object ([ string $class_name [, array $params ]] )
```

Ejemplo:

```
<?php
$mysqli = new mysqli("localhost", "mi_usuario", "mi_contraseña", "world");
/* comprobar la conexión */
if (mysqli_connect_errno()) {
    printf("Falló la conexión: %s\n", mysqli_connect_error());
    exit();
}

$consulta = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($resultado = $mysqli->query($consulta)) {
    /* obtener el array de objetos */
    while ($obj = $resultado->fetch_object()) {
```

```
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* liberar el conjunto de resultados */
    $resultado->close();
}
/* cerrar la conexión */
$mysqli->close();
?>
```

Estilo por procedimientos

Sintaxis:

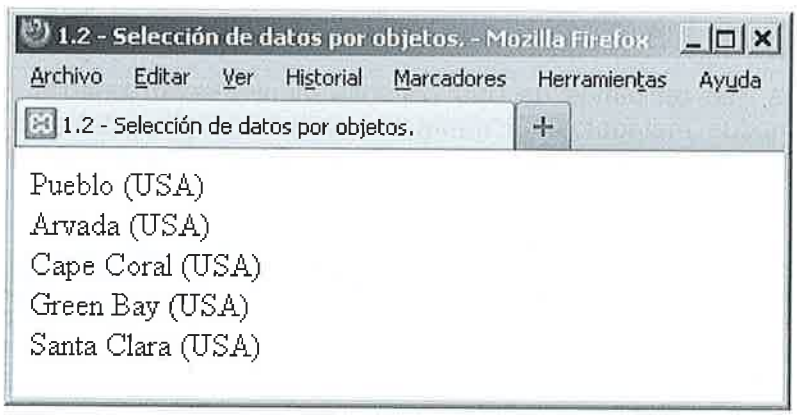
```
object mysqli_fetch_object ( mysqli_result $result [, string $class_name [, array $params ]] )
```

Ejemplo:

```
<?php
$enlace = mysqli_connect("localhost", "mi_usuario", "mi_contraseña", "world");
/* comprobar la conexión */
if (mysqli_connect_errno()) {
    printf("Falló la conexión: %s\n", mysqli_connect_error());
    exit();
}

$consulta = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($resultado = mysqli_query($enlace, $consulta)) {
    /* obtener el array asociativo */
    while ($obj = mysqli_fetch_object($resultado)) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* liberar el conjunto de resultados */
    mysqli_free_result($resultado);
}
/* cerrar la conexión */
mysqli_close($enlace);
?>
```

Teniendo ambos scripts el siguiente resultado en pantalla:



Si en vez de iterar uno a uno todos los resultados de una consulta nos interesa acceder directamente a un determinado registro deberemos utilizar el método de MySQL_Result **data_seek()**. Este recibe un entero que será el número de la fila a la que saltará. Veamos un ejemplo:

```php
<?php
/* Abrir una conexión */
$mysqli = new mysqli("localhost", "mi_usuario", "mi_contraseña", "world");
/* comprobar conexión */
if (mysqli_connect_errno()) {
    printf("Conexión fallida: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = $mysqli->query( $query)) {
    /* saltar a la fila numero 400 */
    $result->data_seek(399);
    /* obtener fila */
    $row = $result->fetch_row();
    printf ("Ciudad: %s  Código de país: %s<br>", $row[0], $row[1]);
    /* librar resultados */
    $result->close();
}
/* cerrar conexión */
$mysqli->close();
?>
```

En el anterior código hemos saltado a la posición 400 de los resultados directamente. En pantalla se mostraría:

Obtener los metadatos.

Los datos que recibimos de una consulta vienen acompañados de unos datos descriptivos llamados metadatos. Si queremos acceder a ellos deberemos el método **fetch_field_direct()** de MySQLi_Result. A continuación se muestra una descripción de los métodos del objeto devuelto.

Atributos del objeto	
Atributo	**Descripción**
name	El nombre de la columna
orgname	El nombre original de la columna si se especificó un alias
table	El nombre de la tabla al que pertenece el campo (si no es calculado)
orgtable	El nombre original de la tabla si se especificó un alias
def	El valor predeterminado de este campo, representado como una cadena
max_length	El ancho máximo del campo del conjunto de resultados.
length	El ancho del campo, como fue especificado en la definición de la tabla.

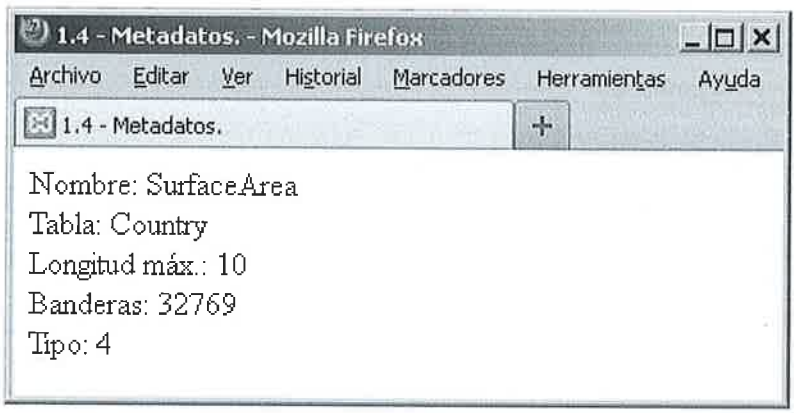
charsetnr	El número del conjunto de caracteres del campo.
flags	Un entero que representa las banderas de bits del campo.
type	El tipo de datos usado por el campo
decimals	El número de decimales usado (para campos integer)

Veamos un ejemplo:

```
<?php
$mysqli = new mysqli("localhost", "mi_usuario", "mi_contraseña", "world");

/* comprobar la conexión */
if (mysqli_connect_errno()) {
    printf("Falló la conexión: %s\n", mysqli_connect_error());
    exit();
}
$consulta = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";
if ($resultado = $mysqli->query($consulta)) {
    /* Obtener la información del campo para la columna 'SurfaceArea' */
    $info_campo = $resultado->fetch_field_direct(1);
    printf("Nombre:       %s<br>", $info_campo->name);
    printf("Tabla:        %s<br>", $info_campo->table);
    printf("Longitud máx.: %d<br>", $info_campo->max_length);
    printf("Banderas:     %d<br>", $info_campo->flags);
    printf("Tipo:         %d<br>", $info_campo->type);
    $resultado->close();
}
/* cerrar la conexión */
$mysqli->close();
?>
```

Teniendo como resultado en pantalla:



Multiconsultas.

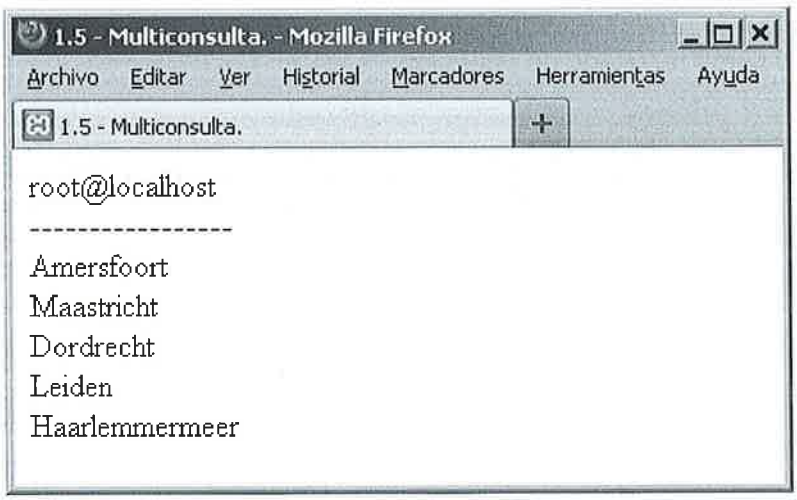
Hay veces que necesitamos ejecutar más de una consulta a la vez y procesar sus datos de forma continuada, en tal caso utilizaremos el método **multi_query()** del objeto MySQLi. Veamos un ejemplo. El siguiente código muestra una primera consulta en la que se muestra el usuario actual con el cual realizamos la consulta y otra consulta para obtener cinco nombres de ciudades a partir de la posición 20.

```
<?php
$mysqli = new mysqli("localhost", "mi_usuario", "mi_contraseña", "world");
```

```
/* comprobar conexión */
if (mysqli_connect_errno()) {
    printf("Conexión fallida: %s\n", mysqli_connect_error());
    exit();
}

$query  = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* ejecutar multi consulta */
if ($mysqli->multi_query($query)) {
    do {
        /* almacenar primer juego de resultados */
        if ($result = $mysqli->store_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->free();
        }
        /* mostrar divisor */
        if ($mysqli->more_results()) {
            printf("-----------------\n");
        }
    } while ($mysqli->next_result());
}
/* cerrar conexión */
$mysqli->close();
?>
```

Por pantalla se obtendría:



Añadiendo registros.

Para añadir registros utilizaremos una estructura muy similar a la utilizada para seleccionar datos, en este caso cambia la instrucción SQL por un INSERT. Veamos un ejemplo para aclarar su uso. En este caso vamos a intentar añadir un nuevo registro en la tabla countrylanguages para el territorio español llamado el "Hispañistaní".

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user", "password", "world");
if ($mysqli === false) {
        die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}
// intenta ejecutar consulta
// añade un nuevo registro
// datos de salida: " Nuevo idioma para España ha sido añadido. "
$sql = "INSERT INTO countrylanguage (CountryCode, Language, IsOfficial, Percentage ) VALUES
('ESP', 'Hispañistaní', true, 69)";
if ($mysqli->query($sql)=== true) {
        echo 'Nuevo idioma para España ha sido añadido.';
} else {
        echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}
// cierra conexión
$mysqli->close();
?>
```

El resultado obtenido por pantalla sería el siguiente:

Si en algún momento necesitamos obtener el id generado tras la ejecución de una consulta INSERT podemos acceder a la propiedad **insert_id**. Veamos un ejemplo:

```
<?php
$mysqli = new mysqli("localhost", "mi_usuario", "mi_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
   printf("Error de conexión: %s\n", mysqli_connect_error());
   exit();
}
$mysqli->query("CREATE TABLE myCity LIKE City");
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
$mysqli->query($query);
printf ("Nuevo registro con el id %d.\n", $mysqli->insert_id);

/* drop table */
$mysqli->query("DROP TABLE myCity");
/* close connection */
$mysqli->close();
?>
```

Modificando registros.

Para modificar los datos de nuestros registros utilizaremos la instrucción SQL UPDATE. Veamos un ejemplo de su uso. En este caso queremos actualizar el uso del idioma "Hispañistaní" y ponerle un porcentaje de uso del 99%.

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user", "password", "world");
if ($mysqli === false) {
        die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}
// intenta ejecutar consulta
// añade un nuevo registro
$sql = "UPDATE countrylanguage SET Percentage = 99 WHERE CountryCode='ESP' and
Language='Hispañistaní'";
if ($mysqli->query($sql)=== true) {
        echo $mysqli->affected_rows . ' fila(s) actualizadas.';
} else {
        echo "ERROR: No fue posible ejecutar: $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Borrando registros.

Exactamente igual que las dos anteriores pero en este caso utilizando la instrucción SQL DELETE. Veamos un ejemplo de su uso. Vamos a borrar el idioma de broma "Hispañistaní".

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "root", "talayuelas", "world");
if ($mysqli === false) {
        die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}
// intenta ejecutar consulta
// añade un nuevo registro
$sql = "DELETE FROM countrylanguage WHERE CountryCode='ESP' and
Language='Hispañistaní'";
if ($mysqli->query($sql)=== true) {
        echo $mysqli->affected_rows . ' fila(s) eliminada(s).';
} else {
        echo "ERROR: No fue posible ejecutar: $sql. " . $mysqli->error;
}
```

```
// cierra conexión
$mysqli->close();
?>
```

Obtener información de la última consulta ejecutada.

La propiedad $info nos devuelve la información de la última consulta ejecutada. Los resultados que puede mostrar son los siguientes:

Posibles valores de retorno de mysqli_info	
Tipo de consulta	**Ejemplo de cadena devuelta**
INSERT INTO...SELECT...	Records: 100 Duplicates: 0 Warnings: 0
INSERT INTO...VALUES (...),(...),(...)	Records: 3 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...	Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE ...	Records: 3 Duplicates: 0 Warnings: 0
UPDATE ...	Rows matched: 40 Changed: 40 Warnings: 0

Veamos un ejemplo de su uso:

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TEMPORARY TABLE t1 LIKE City");
/* INSERT INTO .. SELECT */
$mysqli->query("INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", $mysqli->info);

/* close connection */
$mysqli->close();
?>
```

El resultado de los ejemplos sería: Records: 150 Duplicates: 0 Warnings: 0