MySQL

MySQL es un **S**istema **G**estor de **B**ases de **D**atos (**SGBD**) **relacionales**. Es un programa de código abierto que se ofrece bajo licencia ***GNU*** GPL, aunque también ofrece una licencia comercial en caso de que quieras utilizarlo para desarrollar aplicaciones de código propietario. En las últimas versiones (a partir de la 5.1), se ofrecen, de hecho, varios productos distintos: uno de código libre (Community Edition), y otro u otros comerciales (Standard Edition, Enterprise Edition).

MySQL se emplea en múltiples aplicaciones web, **ligado en la mayor parte de los casos al lenguaje PHP y al servidor web Apache**. Utiliza SQL para la gestión, consulta y modificación de la información almacenada. Soporta la mayor parte de las características de ANSI SQL 99

Herramientas de administración

Existen muchas herramientas que permiten establecer una conexión con un servidor MySQL para realizar tareas de administración. **Algunas herramientas se ejecutan en la línea de comandos, otras presentan un interface gráfico** basado en web o propio del sistema operativo en que se ejecuten. Unas se incluyen con el propio servidor, y otras es necesario obtenerlas e instalarlas de forma independiente. Las hay que están orientadas a algún propósito concreto y también que permiten realizar varias funciones de administración.

Con el servidor MySQL se incluyen algunas herramientas de administración en línea de comandos, entre las que debes conocer:

mysql : Permite conectarse a un servidor MySQL para ejecutar sentencias SQL.

mysqladmin : Es un cliente específico para tareas de administración.

mysqlshow : Muestra información sobre bases de datos y tablas.

En la documentación de MySQL tienes información sobre las distintas utilidades que incorpora: http://dev.mysql.com/doc/refman/5.0/es/client-side-scripts.html

Para **establecer una conexión** al servidor local con la herramienta mysql, podemos hacer: **mysql –u root –p**

Conviene no indicar nunca la contraseña en la misma línea de comandos. En caso de que la cuenta esté convenientemente protegida por una contraseña, es mejor utilizar solo la opción -p como en el ejemplo anterior. De esta forma, la herramienta solicita la introducción de la contraseña y ésta no queda almacenada en ningún registro como puede ser el historial de comandos del sistema.

De entre el resto de herramientas de administración independientes que podemos utilizar con MySQL, podemos destacar dos:

MySQL Workbench es una herramienta genérica con interface gráfico nativo que permite administrar tanto el servidor como las bases de datos que éste gestiona. Ha sido desarrollada por los creadores de MySQL y se ofrece en dos ediciones, una de ellas de código abierto bajo licencia GPL.
http://dev.mysql.com/doc/workbench/en/index.html

phpMyAdmin es una aplicación web muy popular para la administración de servidores MySQL. Presenta un interface web de administración programado en PHP bajo licencia GPL. Su objetivo principal es la administración de las bases de datos y la gestión de la información que maneja el servidor.
http://www.phpmyadmin.net/

Extensión MySQLi.

Esta extensión se desarrolló para aprovechar las ventajas que ofrecen las versiones 4.1.3 y posteriores de MySQL, y viene incluida con PHP a partir de la versión 5. Ofrece un interface de programación dual, pudiendo accederse a las funcionalidades de la extensión utilizando objetos o funciones de forma indiferente. Por ejemplo, para establecer una conexión con un servidor MySQL y consultar su versión, podemos utilizar cualquiera de las siguientes formas:

```
// utilizando constructores y métodos de la programación orientada a objetos
$conexion = new mysqli('localhost', 'usuario', 'contraseña', 'base_de_datos');
print conexion->server_info;
// utilizando llamadas a funciones
$conexion = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

En ambos casos, la variable $conexion es de tipo objeto. La utilización de los métodos y propiedades que aporta la clase mysqli normalmente produce un código más corto y legible que si utilizas llamadas a funciones.

Como ya viste en la primera unidad, las opciones de configuración de PHP se almacenan en el **fichero php.ini** . En este fichero hay una sección específica para las opciones de configuración propias de cada extensión. Entre las opciones que puedes configurar para la extensión MySQLi están:

mysqli.allow_persistent . Permite crear conexiones persistentes.

mysqli.default_port . Número de puerto TCP predeterminado a utilizar cuando se conecta al servidor de base de datos.

mysqli.reconnect . Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.

mysqli.default_host . Host predeterminado a usar cuando se conecta al servidor de base de datos.

mysqli.default_user . Nombre de usuario predeterminado a usar cuando se conecta al servidor de base de datos.

mysqli.default_pw . Contraseña predeterminada a usar cuando se conecta al servidor de base de datos.

Establecimiento de conexiones

Para poder comunicarte desde un programa PHP con un servidor MySQL, el primer paso es establecer una conexión. Toda comunicación posterior que tenga lugar, se hará utilizando esa conexión.

Si utilizas la extensión MySQLi, establecer una conexión con el servidor significa crear una instancia de la clase mysqli. El constructor de la clase puede recibir seis parámetros, todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:

El nombre o dirección IP del servidor MySQL al que te quieres conectar.
Un nombre de usuario con permisos para establecer la conexión.
La contraseña del usuario.
El nombre de la base de datos a la que conectarse.
El número del puerto en que se ejecuta el servidor MySQL.
El socket o la tubería con nombre (named pipe) a usar.

Si utilizas el constructor de la clase, para conectarte a la base de datos "dwes" puedes hacer:

```
// utilizando el constructor de la clase
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
```

Aunque también tienes la opción de primero crear la instancia, y después utilizar el **método connect** para establecer la conexión con el servidor:

```
// utilizando el método connect
$dwes = new mysqli();
$dwes->connect('localhost', 'dwes', 'abc123.', 'dwes');
Por el contrario, utilizando el interface procedimental de la extensión:
// utilizando llamadas a funciones
$dwes = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
```

Es importante **verificar que la conexión se ha establecido correctamente**. Para comprobar el error, en caso de que se produzca, puedes usar las siguientes propiedades (o funciones equivalentes) de la clase mysqli:

connect_errno (o la función mysqli_connect_errno) devuelve el número de error o null si no se produce ningún error.

connect_error (o la función mysqli_connect_error) devuelve el mensaje de error o null si no se produce ningún error.

Por ejemplo, el siguiente código comprueba el establecimiento de una conexión con la base de datos " dwes " y finaliza la ejecución si se produce algún error:

```
@ $dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$error = $dwes->connect_errno;
if ($error != null) {
echo "<p>Error $error conectando a la base de datos: $dwes->connect_error</p>";
exit();
}
```

En PHP, como veremos posteriormente con más detalle, puedes anteponer a cualquier expresión el **operador de control de errores @** para que se ignore cualquier posible error que pueda producirse al ejecutarla (http://es.php.net/manual/es/language.operators.errorcontrol.php)

Si una vez establecida la conexión, quieres **cambiar la base de datos** puedes usar el método **select_db** (o la función mysqli_select_db de forma equivalente) para indicar el nombre de la nueva.

```
// utilizando el método connect
$dwes→select_db('otra_bd');
```

Una vez finalizadas las tareas con la base de datos, utiliza el **método close** (o la función mysqli_close) para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

```
$dwes→close();
```

Ejecución de consultas

La forma más inmediata de ejecutar una consulta, si utilizas esta extensión, es el **método query** , equivalente a la **función mysqli_query** . Si se ejecuta una consulta de acción que no devuelve datos (como una sentencia SQL de tipo UPDATE , INSERT o DELETE), la llamada devuelve true si se ejecuta correctamente o false en caso contrario. El número de registros afectados se puede obtener con la propiedad affected_rows (o con la función mysqli_affected_rows).

```
@ $dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$error = $dwes->connect_errno;
if ($error == null) {
$resultado = $dwes->query('DELETE FROM stock WHERE unidades=0');
if ($resultado) {
print "<p>Se han borrado $dwes->affected_rows registros.</p>";
}
$dwes->close();
}
```

En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto resultado (de la **clase mysqli_result**). En el punto siguiente verás cómo se pueden manejar los resultados obtenidos.

El método query tiene un parámetro opcional que afecta a cómo se obtienen internamente los resultados, pero no a la forma de utilizarlos posteriormente. En la opción por defecto, **MYSQLI_STORE_RESULT** , los resultados se recuperan todos juntos de la base de datos y se almacenan de forma local.

Si cambiamos esta opción por el valor **MYSQLI_USE_RESULT** , los datos se van recuperando del servidor según se vayan necesitando.

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock', MYSQLI_USE_RESULT);
```

Otra forma que puedes utilizar para ejecutar una consulta es el ***método*** **real_query** (o la ***función*** **mysqli_real_query**), que siempre devuelve true o false según se haya ejecutado correctamente o no. Si la consulta devuelve un conjunto de resultados, se podrán recuperar de forma completa utilizando el ***método*** **store_result**, o según vaya siendo necesario gracias al ***método*** **use_result**. (http://es.php.net/manual/es/mysqli.real-query.php)

Es importante tener en cuenta que **los resultados obtenidos se almacenarán en memoria mientras los estés usando**. Cuando ya no los necesites, los puedes liberar con el ***método*** **free** de la clase

```
mysqli_result (o con la función mysqli_free_result ):
$resultado-&gt;free();
```

De las dos opciones que admite el método query, **MYSQLI_STORE_RESULT** y **MYSQLI_USE_RESULT**, ¿qué opción será recomendable utilizar para ejecutar una consulta que devuelva una enorme cantidad de datos? **MYSQLI_USE_RESULT**. Con esta opción se van obteniendo los datos del servidor a medida que se vayan necesitando. Si utilizaras la otra opción, los datos tendrían que transferirse todos juntos al ejecutar la consulta

Obtención y utilización de conjuntos de resultados

Al ejecutar una consulta que devuelve datos obtienes un objeto de la ***clase*** **mysqli_result** . Esta clase sigue los criterios de ofrecer un interface de programación dual, es decir, una función por cada método con la misma funcionalidad que éste.

Para trabajar con los datos obtenidos del servidor, tienes varias posibilidades:
fetch_array (***función*** **mysqli_fetch_array**) . Obtiene un registro completo del conjunto de resultados y lo almacena en un array. Por defecto el array contiene tanto claves numéricas como asociativas.

Por **ejemplo**, para acceder al primer campo devuelto, podemos utilizar como clave el número 0 o su nombre indistintamente.

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock WHERE unidades<2');
$stock = $resultado->fetch_array(); // Obtenemos el primer registro
$producto = $stock['producto']; // O también $stock[0];
$unidades = $stock['unidades']; // O también $stock[1];
print "<p>Producto $producto: $unidades unidades.</p>";
```

Este comportamiento por defecto se puede modificar utilizando un parámetro opcional, que puede tomar los siguientes valores:

MYSQLI_NUM . Devuelve un array con claves numéricas.

MYSQLI_ASSOC . Devuelve un array asociativo.

MYSQLI_BOTH . Es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.

- **fetch_assoc** (función mysqli_fetch_assoc) . Idéntico a fetch_array pasando como parámetro MYSQLI_ASSOC .
- **fetch_row** (función mysqli_fetch_row) . Idéntico a fetch_array pasando como parámetro MYSQLI_NUM .
- **fetch_object** (función mysqli_fetch_object) . Similar a los métodos anteriores, pero devuelve un objeto en lugar de un array. Las propiedades del objeto devuelto se corresponden con cada uno de los campos del registro.

Parar recorrer todos los registros de un array, puedes hacer un bucle teniendo en cuenta que cualquiera de los métodos o funciones anteriores devolverá null cuando no haya más registros en el conjunto de resultados.

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock WHERE unidades<2');
$stock = $resultado->fetch_object();
while ($stock != null) {
print "<p>Producto $stock->producto: $stock->unidades unidades.</p>";
$stock = $resultado->fetch_object();
}
```

Consultas preparadas

Cada vez que se envía una consulta al servidor, éste debe analizarla antes de ejecutarla. Algunas sentencias SQL, como las que insertan valores en una tabla, deben repetirse de forma habitual en un programa. Para acelerar este proceso, MySQL admite consultas preparadas. **Estas consultas se almacenan en el servidor listas para ser ejecutadas cuando sea necesario.**

Para trabajar con consultas preparadas con la extensión MySQLi de PHP, debes utilizar la **clase mysqli_stmt**. Utilizando el ***método* stmt_init** de la clase mysqli (o la ***función* mysqli_stmt_init**) obtienes un objeto de dicha clase.

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$consulta = $dwes→stmt_init();
```

Los pasos que debes seguir para ejecutar una consulta preparada son:

Preparar la consulta en el servidor MySQL utilizando el ***método* prepare** (función mysqli_stmt_prepare).

Ejecutar la consulta, tantas veces como sea necesario, con el ***método* execute** (función mysqli_stmt_execute).

Una vez que ya no se necesita más, se debe ejecutar el ***método* close** (función mysqli_stmt_close).

Por **ejemplo**, para preparar y ejecutar una consulta que inserta un nuevo registro en la tabla familia:

```
$consulta = $dwes->stmt_init();
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES ("TABLET", "Tablet PC")');
$consulta->execute();
$consulta->close();
$dwes→close();
```

El problema que ya habrás observado, es que de poco sirve preparar una consulta de inserción de datos como la anterior, si los valores que inserta son siempre los mismos. Por este motivo las consultas preparadas admiten parámetros.

Para preparar una consulta con parámetros, en lugar de poner los valores debes indicar con un signo de interrogación su posición dentro de la sentencia SQL.

```
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
```

Y antes de ejecutar la consulta tienes que utilizar el ***método* bind_param** (o la función mysqli_stmt_bind_param) para sustituir cada parámetro por su valor. El primer parámetro del método bind_param es una cadena de texto en la que cada carácter indica el tipo de un parámetro, según la siguiente tabla.

Caracteres indicativos del tipo de los parámetros en una consulta preparada	
Carácter	Tipo del parámetro
I	Número entero
D	Número real (doble precisión)
S	Cadena de texto
B	Contenido en formato binario (BLOB)

En el caso anterior, si almacenas los valores a insertar en sendas variables, puedes hacer:

```
$consulta = $dwes->stmt_init();
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
$cod_producto = "TABLET";
$nombre_producto = "Tablet PC";
$consulta->bind_param('ss', $cod_producto, $nombre_producto);
$consulta->execute();
$consulta->close();
$dwes→close();
```

Cuando uses bind_param para enlazar los parámetros de una consulta preparada con sus respectivos valores, deberás usar siempre variables como en el ejemplo anterior. Si intentas utilizar literales, por **ejemplo**:

```
$consulta->bind_param('ss', 'TABLET', 'Tablet PC'); // Genera un error
```

Obtendrás un error. El motivo es que los parámetros del método bind_param se pasan por referencia. Aprenderás a usar paso de parámetros por referencia en una unidad posterior.

El ***método*** **bind_param permite tener una consulta preparada en el servidor MySQL y ejecutarla tantas veces como quieras cambiando ciertos valores cada vez**. Además, en el caso de las consultas que devuelven valores,
se puede utilizar el ***método*** **bind_result** (función mysqli_stmt_bind_result) **para asignar a variables los campos que se obtienen tras la ejecución**.

Utilizando el método fetch (mysqli_stmt_fetch) se recorren los registros devueltos. Observa el siguiente código:

```
$consulta = $dwes->stmt_init();
$consulta->prepare('SELECT producto, unidades FROM stock WHERE unidades<2');
$consulta->execute();
$consulta->bind_result($producto, $unidades);
while($consulta->fetch()) {
print "<p>Producto $producto: $unidades unidades.</p>";
}
$consulta->close();
$dwes->close();
```