Unidad 7

# Acceso a base de datos desde lenguajes de script de servidor

## Implantación de aplicaciones web

# Índice

# Introducción

A la hora de crear una página web dinámica se deben de tener implementados en el servidor web, tanto un lenguaje de script del servidor, en este caso PHP, como un sistema gestor de bases de datos para que la información que el sitio web genere se almacene adecuadamente, para su posterior gestión. Cuando usamos una base de datos en un sitio web podemos realizar las siguientes acciones:

- Establecer mecanismos de control de acceso a la web.
- Guardar las preferencias de los usuarios.
- Ofrecer información actualizada con frecuencia.
- Buscar información.

Todo lo anterior hace totalmente imprescindible usar una base de datos en cualquier implementación web actual.

# Al finalizar esta unidad

- Sabremos como aprender el esquema general de acceso a bases de datos desde un lenguaje de script.
- Sabremos conectar desde PHP a un sistema gestor de bases de datos.
- Conoceremos los comandos que permiten seleccionar bases de datos desde PHP.
- Conoceremos los comandos para poder crear una tabla desde una base de datos accedida desde PHP.
- Podremos lanzar consultas de selección sobre las tablas de una base de datos.
- Seremos capaces de ejecutar sentencias de actualización sobre una o más tablas, sentencias de eliminación sobre una o más tablas y sentencias de inserción.
- Conoceremos los fundamentos de seguridad de un sitio web.

# 7.1. Bases de datos en la web

Existen numerosos lenguajes de *script* que funcionan en el servidor, pero independientemente del que usemos, siempre debe de integrarse con un SGBD, siguiendo por lo general la siguiente secuencia de pasos:

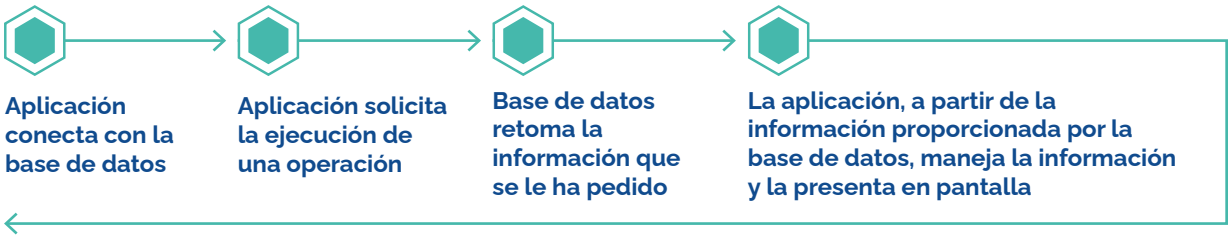


Imagen 1. Integración lenguaje *script* y base de datos

PHP se puede usar para acceder a distintos SGBD, ya sean libres o propietarios, pero dependiendo de a cuál queramos acceder, usaremos unos comandos u otros. Nosotros vamos a ver a continuación como conectar con *MySQL*.

# 7.2. Acceso a MySQL desde PHP

En este apartado vamos a dar como sabido que se ha hecho una instalación de la base de datos pertinente y de PHP, por lo que directamente veremos conectar. En casi todos los ámbitos, la conexión a *MySQL* mediante PHP seguirá la siguiente estructura:

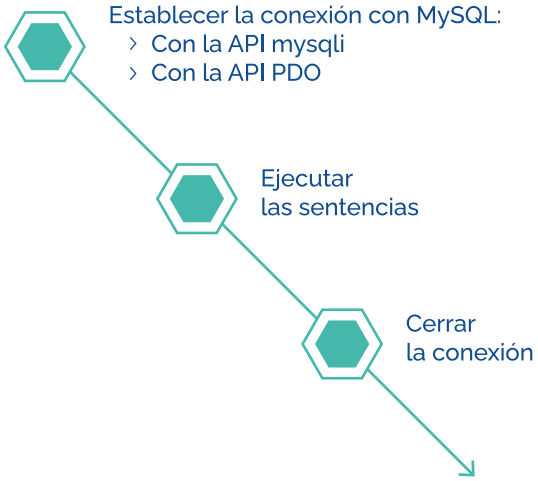


Imagen 2. Acceder a MySQL desde PHP

## 7.2.1. Conectar con un servidor MySQL base de datos desde PHP

Si queremos acceder a MySQL usando PHP, tenemos las siguientes API:

- **MySQLi (MySQL improved):** es una versión mejorada del controlador MySQL que permite que se acceda a una base de datos usando PHP.
- **PDO (PHP data object, objeto de datos PHP):** esta API permite todo lo que nos permite *MySQLi*, pero añade además que se puedan conectar diferentes tipos de bases de datos relacionales. Por ejemplo, podríamos además de con MySQL, conectar con una base de datos *PostGreSQL*.

En el siguiente cuadro podemos ver los principales usos de sintaxis para crear nuevas conexiones a MySQL mediante PHP usando *MySQLi*:

| | |
|---|---|
| **Sintaxis** | `$variable = new mysqli (“servidor”,”usuario”,”contraseña”, ”base_de_datos”,puerto);` |
| **Parámetros de conexión** | + **SERVIDOR:** es el nombre del servidor MySQL al que deseamos conectarnos, se puede indicar o el nombre del servidor o la dirección IP. |
| | + **USUARIO:** es el nombre del usuario que se usa para iniciar sesión en el SGBD. |
| | + **CONTRASEÑA:** contraseña del usuario anterior. |
| | + **BASE_DE_DATOS:** la base de datos a la que queramos conectarnos. |
| | + **PUERTO:** esta parte es opcional, pero es el puerto de escucha de MySQL. Por defecto, si no lo ponemos se coge el 3306. |

Vamos a ver ahora un pequeño ejemplo de *script* del servidor. Por ejemplo, simplemente vamos a probar la conexión a MySQL.

El código quedaría del siguiente modo:

```
 GNU nano 6.2                                                    conexion.php
<?php
$conexion =  new mysqli("localhost","asir","universae","asir","3360");
if ($conexion->connect_errno) {
        echo "Fallo al conectar a MySQL: (" . $conexion->connect_errno . ") " . $conexion->connect_error;
}
echo $conexion ->host_info . "\n";
?>
```

Imagen 3. *conexion.php*

Este código lo que hace es lo siguiente:

1. Primero crea la variable *conexion* con los parámetros que hemos visto anteriormente en el cuadro de sintaxis.
2. En segunda instancia crea una condición, en la que, si la variable da error, muestra el mensaje de error de la conexión.
3. Por último, muestra la información del servidor de la base de datos.

Si ejecutamos el *script* en el mismo servidor, vemos que nos muestra la información como hemos dicho anteriormente.

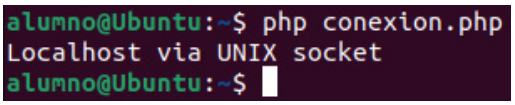


Imagen 4. Ejecución de *conexion.php* en terminal

Se puede alojar este fichero en el directorio de páginas de apache y si se ejecuta en un navegador, se comprueba que la salida también es la misma.

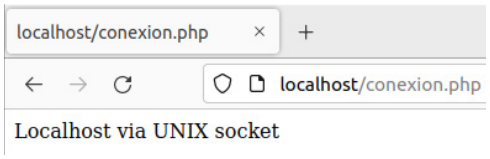


Imagen 5. Ejecución de *conexion.php* en navegador web

Hay ocasiones en las que la sentencia *echo* no es suficiente para mostrar información en un navegador web, por lo que tendremos que indicar la información con la sentencia *print*. Es más, puede incluso que haya ocasiones en los que esto no sea suficiente y tendríamos que incrustar el código PHP en código HTML.

## 7.2.2. Controlar la conexión

Antes hemos visto en el código ciertas instrucciones que indicaban si la conexión era correcta o no, es ahora el momento de explicar estos controles de conexión.

Es importante que en las bases de datos se controle la conexión y su fiabilidad además de saber que se está haciendo un uso correcto de esta. Para controlar todo esto, existen una serie de instrucciones que ayudan a conocer los distintos errores que dan las bases de datos y para finalizar correctamente la conexión la base de datos.

En el siguiente cuadro tenemos un resumen de los métodos que se usan para asegurar que no se tenga una conexión persistente con las bases de datos.

| | |
|---|---|
| **Cerrar conexión con la base de datos** | `$variable->close();` |
| **Control de errores** | + **connect_error:** método booleano que indica si hay o no error. |
| | + **connect_errno:** método que indica que número de error es en caso de que lo haya. |
| | + **error_list:** método que muestra la lista de todos los errores que suceden durante una conexión. |

La función *close()* se usa para cerrar la conexión con la base de datos, algo casi imprescindible en temas de seguridad.

## 7.2.3. Seleccionar bases de datos

Puede que, aunque hayamos establecido ya una conexión con la base de datos tal y como queremos, queramos conectarnos a otra en un momento especifico de la conexión. Para realizar dicha acción debemos de usar el método *select_db()*. Este es un método *booleano* que si funciona la conexión con la nueva base de datos nos devolverá *true* y si no se puede conectar nos devolverá *false*.

## 7.2.4. Ejecutar consultas SQL

### Secuencia de ejecución de consultas

Cuando ya se ha establecido la conexión con el servidor de base de datos y además se ha seleccionado la base de datos que queremos, es el momento de enviar consultas a MySQL para procesarlas. La función usada en este momento es *query*, ya que permite que se ejecuten sentencias SQL literalmente. Si la función se puede ejecutar y devuelve *true*, puede devolver además otros resultados dependiendo del tipo de sentencia que se haya ejecutado. Tenemos los siguientes resultados como posibilidades:

- Para las sentencias que devuelven un conjunto de resultados, se retorna un objeto tipo *mysqli_result*. La sentencia *SELECT* tiene un tratamiento especial, ya que lo que se obtiene como conjunto de resultados se denominas *result set*, que es una estructura de resultados en forma de tabla que se recoge en un *array asociativo*. Este tipo de array es una estructura de datos donde cada índice corresponde al nombre de cada campo, correspondiendo los valores a la primera fila del resultado de la instrucción lanzada. Si queremos recorrer cada una de las filas habrá que usar un método que se denomina *fetch_assoc* junto a una instrucción repetitiva de PHP.
- Para las demás sentencias se devuelve *true* si la ejecución ha sido satisfactoria y *false* en caso de fallo.

El siguiente cuadro nos muestra la sintaxis de la función *query* junto a la secuencia general de pasos para ejecutar una sentencia SQL.

| | |
|---|---|
| **Sintaxis de la función query** | `$variable->query("sentencia SQL");` |
| **Secuencia de ejecución** | + Abrir la conexión con la base de datos. |
| | + Escribir la sentencia SQL |
| | + Ejecutar la consulta SQL usando la función *query*. |
| | + Si se retorna un conjunto de valores después de realizar la consulta, crear un proceso para poder mostrar los datos en un navegador web. |
| | + Cerrar la conexión con la base de datos. |

Tenemos además dos funciones que se usan mucho junto con la función *query* para tratar la información obtenida de los resultados:

- **num_rows:** retorna el número de filas que devuelve la ejecución de una sentencia *SELECT*.
- **data_seek:** permite colocarse en un número de fila determinado cuando se obtiene un conjunto de resultados.

### Crear base de datos de MySQL desde PHP

Si ya sabemos trabajar con bases de datos, con PHP podemos crear bases de datos en MySQL. Para crear una base de datos debemos realizar los siguientes pasos:

1. Crear un *script* en PHP que conecte con la base de datos y recoja una función *query* con la sentencia *CREATE DATABASE*.

```
 GNU nano 6.2                                  base_datos.php
<?php
//Primero nos conectamos a la base de datos
$server = "localhost";
$user = "asir";
$password = "universae";

//Abrimos la conexión con el SGBD
$conexion = new mysqli($server,$user,$password);

//Comprobamos la conexión
if ($conexion->connect_error) {
        echo "La conexión ha fallado: " . $conexion->connect_error;
        $conexion->close();
        echo "\n";
} else
{
        echo "La conexión ha funcionado";
        echo "\n";
}

//Vamos a crear la base de datos
$bbdd = "CREATE DATABASE prueba";
if ($conexion->query($bbdd) === TRUE) {
        echo "Se ha creado la base de datos";
        echo "\n";
        $conexion->close();
}

?>
```

Imagen 6. *base_datos.php*

Podemos observar que no hay un control de errores en caso de que no se pueda crear la base de datos, esto es debido a que realmente requiere de un nivel más alto de control de PHP que no se va a ver en esta asignatura. Si falla, simplemente se saldrá del *script* y tendremos que cerrar la conexión manualmente.

2. Ejecutamos el *script*.

```
alumno@Ubuntu:~$ php base_datos.php
La conexión ha funcionado
Se ha creado la base de datos
alumno@Ubuntu:~$
```

Imagen 7. Ejecución del *script* para crear la base de datos

3. Comprobamos que se ha creado correctamente la base de datos.

```
alumno@Ubuntu:~$ sudo mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 24
Server version: 8.0.30-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| asir               |
| information_schema |
| mysql              |
| performance_schema |
| prueba             |
| sys                |
+--------------------+
6 rows in set (0,00 sec)

mysql>
```

Imagen 8. Comprobación de que la base de datos se ha creado

## Crear tablas en una base de datos de MySQL desde PHP

Para crear tablas en MySQL con PHP, lanzamos un proceso muy similar al anterior, pero con pequeños cambios.

Lo primero es que hay que seleccionar la base de datos donde se va a crear la tabla, ya sea en la conexión o con una sentencia posterior.

Luego, dentro de la función *query* agregamos la sentencia *CREATE TABLE tabla(campos)*.

Un ejemplo podría ser:

```php
<?php
//Primero nos conectamos a la base de datos
$server = "localhost";
$user = "asir";
$password = "universae";
$base_datos = "prueba";

//Abrimos la conexión con el SGBD
$conexion = new mysqli($server,$user,$password,$base_datos);

//Comprobamos la conexión
if ($conexion->connect_error) {
        echo "La conexión ha fallado: " . $conexion->connect_error;
        $conexion->close();
        echo "\n";
} else
{
        echo "La conexión ha funcionado";
        echo "\n";
}

//Vamos a crear la tabla
$bbdd = "CREATE TABLE prueba_tabla(idprueba INT(4))";
if ($conexion->query($bbdd) === TRUE) {
        echo "Se ha creado la tabla";
        echo "\n";
        $conexion->close();
}

?>
```

Imagen 9. tabla.php

Comprobamos que se ha creado la tabla correctamente.

```
mysql> use prueba;
Reading table information for completion of
You can turn off this feature to get a quick

Database changed
mysql> show tables;
+-----------------+
| Tables_in_prueba |
+-----------------+
| prueba_tabla     |
+-----------------+
1 row in set (0,00 sec)

mysql>
```

Imagen 10. Comprobamos que la tabla se ha creado

## Ejecución de sentencias de modificación desde PHP

Si queremos cambiar los valores de distintos campos de una tabla que se ha creado en nuestra base de datos.

Para ejecutar cualquiera de las sentencias que modifican los cambios de una tabla se deberá de usar la sentencia correspondiente junto a una instrucción iterativa para cambiar el de varias filas, si no, el proceso será similar al anteriormente visto.

# 7.3. Mecanismos de seguridad y control de acceso

Cualquier página web que se crea y se publica en internet es susceptible de recibir ataques, lo que hace que sea necesario implementar mecanismos de seguridad que aseguren todo lo posible la información almacenada.

Existen una serie de sentencias de SQL que malintencionadamente pueden dañar el sistema, y son las llamadas inyecciones SQL. Una inyección SQL se puede definir como:

El INCIBE, el *Instituto Nacional de CiberSeguridad* de España, https://www.incibe-cert.es/, dice que los ataques mediante inyecciones SQL es uno de los diez principales riesgos críticos de seguridad de las aplicaciones web. Los riesgos los publica el INCIBE de manera anual en su página web en forma de informe. Dicho informe se llama *OWASP Top 10*.

Es casi imposible controlar todas las amenazas que puede sufrir una aplicación web, por lo que lo importante a la hora de la verdad es dotar a nuestro sistema de la seguridad más robusta posible.

En el siguiente diagrama podemos ver de manera muy resumida los principales aspectos de seguridad a considerar en los sitios web:


Imagen 11. Seguridad en los sitios web

### 7.3.1. Algunas medidas de seguridad que deben tomarse en PHP

Como se ha dicho en múltiples ocasiones en esta y otras asignaturas, una de las mayores medidas de seguridad que se debe tomar es mantener todos los servicios y lenguajes actualizados en nuestro servidor. Complementario a esto, podemos tomar también las siguientes precauciones:

- Configurar el fichero *php.ini* de manera eficiente y segura.
- Aceptar solo datos que sean conocidos como válidos.
- Validar estos datos para asegurarnos de su veracidad.

### 7.3.2. Control de accesos en PHP

Ya hemos hablado de la importancia de la información que se introduce en un sitio web en cuanto a seguridad se refiere. PHP permite que se establezcan mecanismos de validación de los datos mandándolos a un fichero. Dicho fichero tendrá que ser capaz de procesar y validar los datos del usuario que intente *loguearse* a nuestro sitio web mediante el acceso a una base de datos, por ejemplo, MySQL. Cada vez que un usuario se intente conectar se le asignará un identificador único para tener control sobre dicho usuario.

Además, los sitios web suelen tener numerosos formularios donde se introducirán datos que sean comunes, por lo que cuando un usuario hace *login*, es muy interesante conservar los valores que se introducen en el proceso de verificación de credenciales.

Cuando una página implementada con PHP quiere acceder a los datos de usuario para enviarlos a otra página, usará los siguientes dos métodos:

- **Cookies:** se almacenan en el cliente.
- **Sesiones:** se almacenan en el servidor.

Para poder almacenar las *cookies* es necesario que el navegador tenga esto habilitado. Si esto es así, PHP contiene una función llamada *setcookie()* que crea una *cookie* para enviarla junto al resto de cabeceras HTTP.

Otro mecanismo que puede ser implementado con PHP durante la navegación son los datos que se recogen en la sesión. Estas sesiones usan las *cookies* para conservar los valores identificativos de los usuarios mientras viaja a diferentes páginas, ejemplo de esto es cuando añadimos nuestra cuenta de Google en *Google Chrome*. Las principales diferencias entre *cookies* y sesiones son las siguientes:

- **Ubicación.** Da igual que el usuario deshabilite las *cookies* del navegador, porque la información de la sesión se almacena en el servidor.
- **Persistencia.** A las sesiones solo se puede acceder mientras la navegación esté activa, al menos en la mayoría de los casos, mientras que las *cookies* se quedan almacenadas en el navegador a no ser que se eliminen manualmente.

www.universae.com