# Desarrollo web en entorno servidor

## Unidad 6: Utilización de técnicas de acceso a datos

## ÍNDICE

# INTRODUCCIÓN

Aunque la finalidad y la diversidad de aplicaciones que existen es difícilmente clasificable, sí puede decirse que todas tienen en común unos componentes básicos: una **lógica de negocio** que determinará la función del software, las **conexiones con sistemas externos**, una serie de **interfaces de usuario** y los **datos que deben ser guardados en bases de datos** con la capacidad de ser modificados.

La diversidad de aplicaciones está relacionada con la diversidad de bases de datos (BD) que existen, y en función de la aplicación que está construyendo tendrá que escoger una base de datos u otra.


Esto podría presentar un problema si para cada tipo de base de datos debería escribir un código diferente. En esta unidad didáctica aprenderemos primero a trabajar con la base de datos MySQL pero también veremos que existen mecanismos para trabajar en múltiples orígenes de datos.

DESTACADO *En este curso nos centraremos en el uso de PHP con motores de base de datos MySQL.*

## *OBJETIVOS / CAPACIDADES*

***En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:***

- ✔ **Utilizar lenguajes, objetos y herramientas, interpretando las especificaciones para desarrollar aplicaciones web con acceso a bases de datos.**
- ✔ **Seleccionar lenguajes, objetos y herramientas, interpretando las especificaciones para desarrollar aplicaciones web con acceso a bases de datos.**
- ✔ **Generar componentes de acceso a datos, cumpliendo las especificaciones, para integrar contenidos en la lógica de una aplicación web.**
- ✔ **Utilizar herramientas y lenguajes específicos, cumpliendo las especificaciones, para desarrollar e integrar componentes software en el entorno del servidor web.**
- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**

# PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

**Enunciado**:

**Almacenando datos de empleados**

Siguiendo con el hilo conductor de las otras actividades de evaluación, ahora queremos almacenar la información de los empleados de nuestra empresa en una base de datos.

La pantalla principal de la aplicación debe mostrar el listado con sólo el nombre y apellidos del empleado/a. Cada registro debe tener 3 opciones, en forma de botones o enlaces:

- **Ver**: debe mostrar, en una nueva página, por ejemplo, nombre, apellidos, edad, cargo y sueldo bruto anual.
- **Editar**: se deben poder editar todos los campos, excepto el identificador del empleado.
- **Eliminar**: se debe poder eliminar el empleado.

También es necesario tener un botón en la pantalla principal para poder añadir un nuevo empleado.

Para realizar el juego de pruebas, crea un mínimo de 3 empleados con todos sus datos. Utiliza la estructura de base de datos que consideres más apropiada.

# 1. UTILIZACIÓN DE BASES DE DATOS RELACIONALES

**PHP** puede acceder a prácticamente cualquier **base de datos**: Oracle, Microsoft Access, MySQL, etc. Pero para el motor **MySQL** tiene **soporte nativo**, lo que indica que su rendimiento será superior con esta base de datos.

Esto, combinado con que MySQL sea un servidor de base de datos de libre distribución, hace que la **combinación PHP-MySQL** sea muy extendida. De hecho, una gran solución para el desarrollo de aplicaciones sin coste de licencia es la combinación llamada **LAMP** (Linux-Apache-MySQL-PHP). Existen diferentes **herramientas** y formas de crear, definir y modificar nuestras bases de datos:


- **MySQL Administrator**.

  Herramienta gráfica que nos permite **conectarnos al motor MySQL** y crear bases de datos, tablas, añadir registros, etc.

- **PHPMyAdmin**.

  Herramienta web que nos permite **gestionar las bases de datos** de nuestro servidor MySQL. A diferencia de lo anterior no nos permite conectarnos a cualquier servidor MySQL. Si trabajamos en un servidor local, por ejemplo, accederemos a él: http: //localhost/phpmyadmin

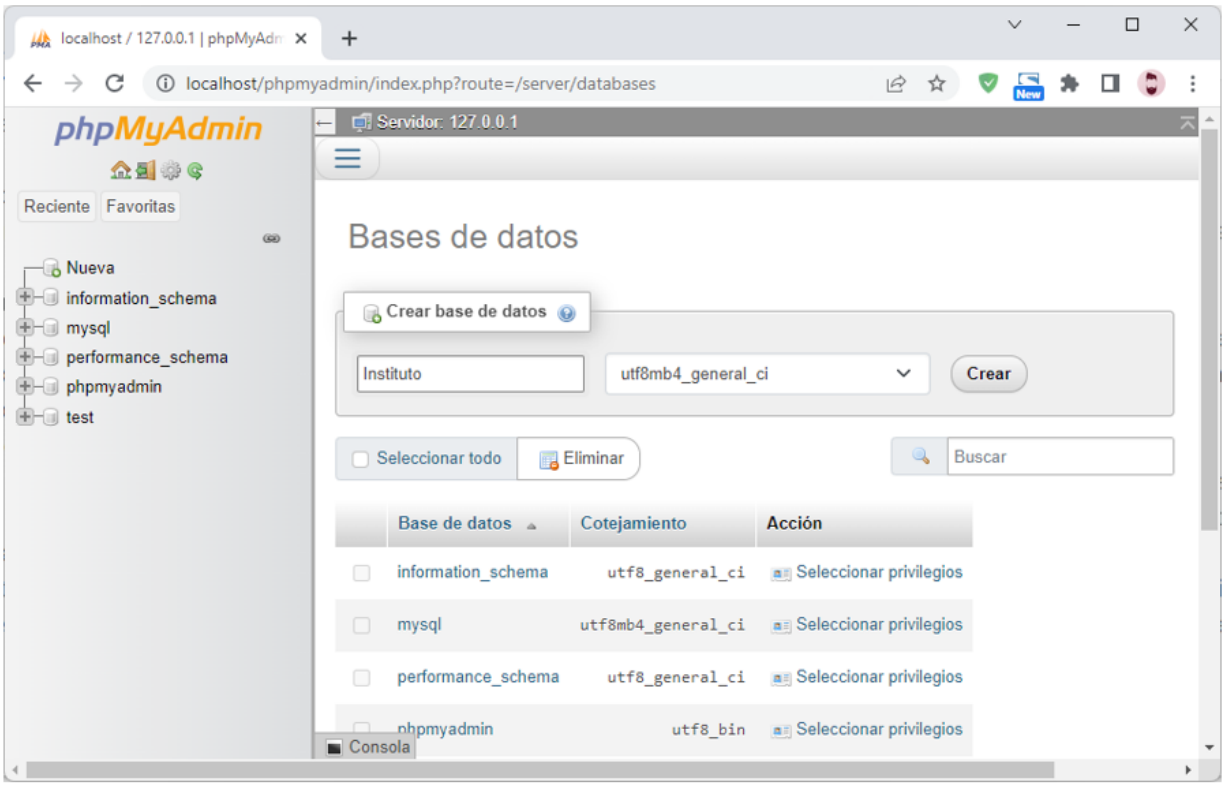
- **SQL**.

  Uso del lenguaje SQL. Normalmente accedemos a nuestras bases de datos desde la línea de comandos con la herramienta mysql (MySQL monitor).

  Nosotros nos centraremos en la conexión, inserción y modificación de la base de datos y no tanto en su diseño. Por eso, aprovechando que trabajamos sobre XAMPP, utilizaremos **PHPMyAdmin**.

Los siguientes apartados los estudiaremos a partir de **ejemplos de una base de datos real** que veremos a lo largo de la unidad por pasos.

➔ **Paso 1**: por ello, nos dirigimos a http://localhost/phpmyadmin y creamos la base de datos **Instituto** tal y como se muestra en la siguiente imagen:

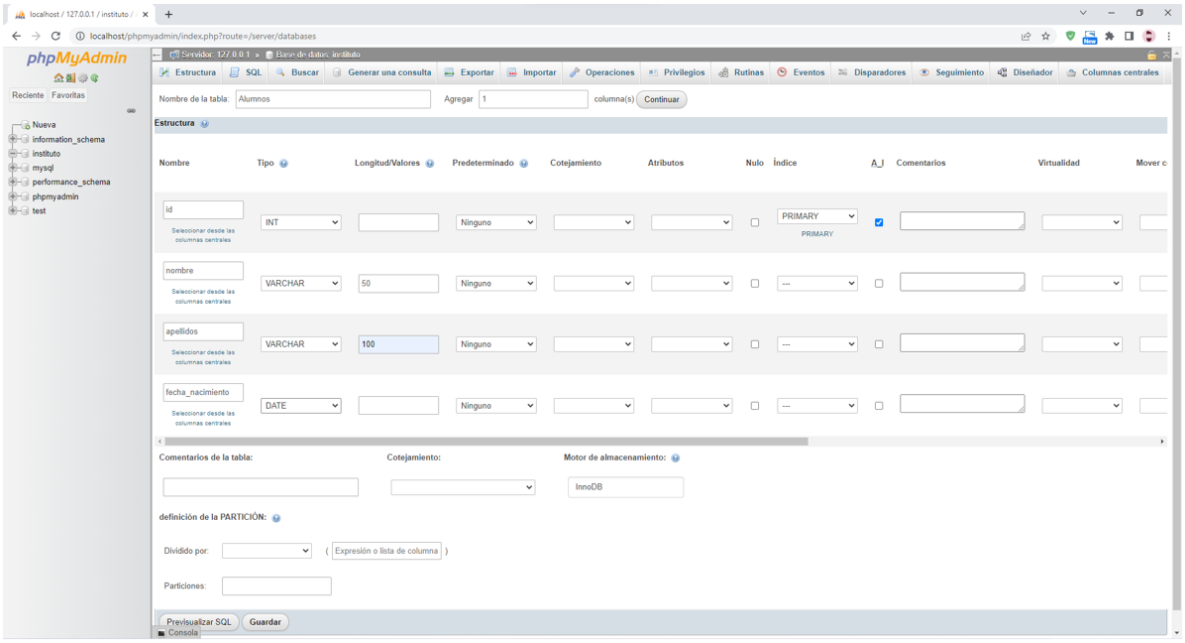



*Creación de una nueva base de datos.*

TOMA NOTA

*Si no hemos creado ningún usuario ni contraseña (opción por defecto) entraremos directamente al **gestor** de phpMyAdmin.*

➔ **Paso 2**: a continuación, creamos la tabla **Alumnos**, con los campos id, nombre, apellidos y fecha_nacimiento, tal y como se observa en la siguiente imagen:

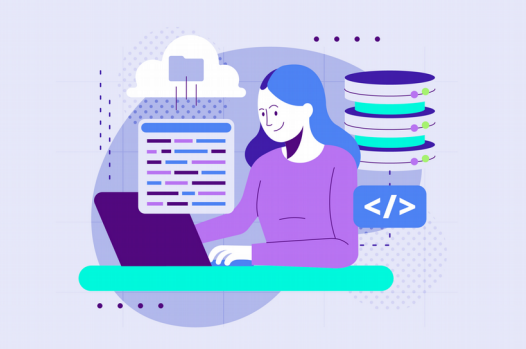



*Creación de una nueva tabla.*


*Es costumbre utilizar **mayúsculas** al inicio del nombre de tablas. Te darás cuenta de que, aunque MySQL lo permite desde la CLI, al utilizar phpMyAdmin puede que lo cambie a minúsculas en función de la configuración. Esto no debe preocuparte.*

## 2. ESTABLECIMIENTO Y CIERRES DE CONEXIONES. CONTROL DE ERRORES

PHP nos ofrece dos herramientas para **conectarnos a bases de datos**:



- **MySQLi extension** (antiguamente conocida como MySQL extension, sin la "i"): ideal para trabajar con bases de datos con motor MySQL.
- **PDO (PHP Data Objects)**: similar a MySQLi pero acepta hasta 12 motores de bases de datos diferentes.

Llegados a este punto, cabe hacerse una pregunta relativa a estas dos opciones: **¿cuál de estas dos herramientas es mejor?** Pues lo cierto es que ambas funcionan de forma **similar**, pero hay algunas cuestiones que tener en cuenta:

DESTACADO

*La ventaja de **PDO**, como hemos dicho, es que trabaja con distintas BD y esto puede ser útil en algunos proyectos. La ventaja de **MySQLi** es que permite trabajar orientado a objetos o de forma procedimental. Dicho esto, cabe destacar que, de momento, trabajaremos con MySQLi de forma **procedimental**.*

Dicho esto, veamos a continuación cómo haremos para **conectarnos con la base de datos** que hemos creado en el punto anterior:

*Conexión con la base de datos - Paso 3*

Retomemos los pasos trabajados en el apartado anterior, en el que comenzamos a trabajar sobre una base de datos real. En este caso, crearemos un nuevo archivo en la carpeta pública de nuestro servidor web. Lo llamamos, por ejemplo, **conexionBD.php** y escribimos el siguiente código:

```php
<?php
	$servername = "localhost";
	$username = "root";
	$password = "";
	$dbname = "Instituto";

	// Creamos una conexión y la guardamos en la variable $conn
	$conn = mysqli_connect($servername, $username, $password, $dbname);

	// Verificamos la conexión
	if (!$conn) {
		die("Error en la conexión: " . mysqli_connect_error());
	}
	echo "Conexión realizada";
?>
```

Si ejecutamos este script y todos los parámetros son correctos, debería salirnos por pantalla el texto "conexión realizada". En caso contrario, si se produce un error, deberíamos ver el texto "error en la conexión:" seguido de la descripción del error.

Si no ves nada en pantalla revisa que el **usuario sea root** y si has puesto o no contraseña en el momento de la instalación de MySQL.

## 3. RECUPERACIÓN Y EDICIÓN DE INFORMACIÓN

En el punto anterior ya hemos creado una conexión con la base de datos. Ahora, vamos a probar a **crear un nuevo registro**. Continuemos avanzando por nuestro proceso paso a paso:


➔ **Paso 4**.

Para insertar un nuevo registro en la base de datos crearemos un nuevo archivo en la carpeta pública llamado **insercionBD.php** con el siguiente código:

```
<?php
      require("conexionBD.php");


      if (!isset($_GET['nombre']) || !isset($_GET['apellidos']) || !
isset($_GET['fecha_nacimiento'])) {
            exit;
      }


      $sql = "INSERT INTO `alumnos` (`nombre`, `apellidos`,
`fecha_nacimiento`) VALUES ('" . $_GET['nombre'] . "', '" .
$_GET['apellidos'] . "', '". $_GET['fecha_nacimiento'] . "')";


      if (mysqli_query($conn, $sql)) {
      echo "Se ha añadido un nuevo alumno.";
      } else {
          echo "Error: " . $sql . "<br>" . mysqli_error($conn);
      }


?>
```

Este script intenta leer los parámetros "nombre", "apellidos" y "fecha_nacimiento" vía GET, y si existen crea una sentencia SQL para insertar un nuevo valor.

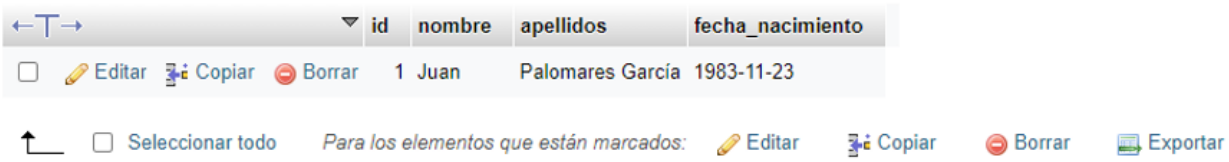
➔ **Ejecución del script**:

Si, por ejemplo, ejecutamos el script de la siguiente forma:

```
http://localhost/instituto/insercionBD.php?
nombre=Juan&apellidos=Palomares García&fecha_nacimiento=1983-11-23
```

➔ **Resultado**:

Y miramos que tenemos ahora en la tabla **alumnos** de nuestra base de datos **instituto** a través de phpMyAdmin, obtendremos algo parecido a:

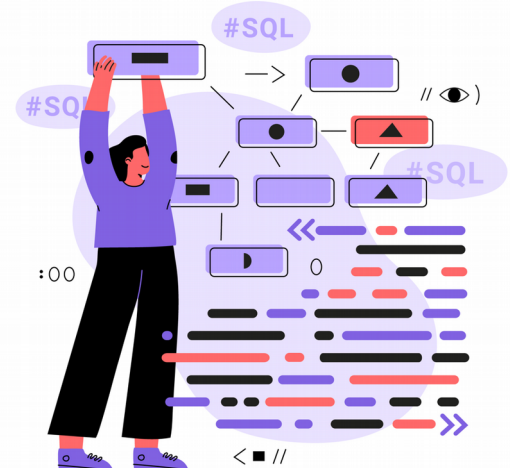



*Resultado de la inserción de un alumno a la base de datos*

# 4. UTILIZACIÓN DE CONJUNTOS DE RESULTADOS

Imaginemos que ya hemos usado el script **insercionBD.php** que creamos en el apartado anterior unas cuantas veces y ya tenemos toda una serie de alumnos dentro de nuestra base de datos.

Vamos a aprender ahora cómo realizar la **consulta** para recuperarlos todos y mostrarlos por pantalla.




➔ **Paso 5**:

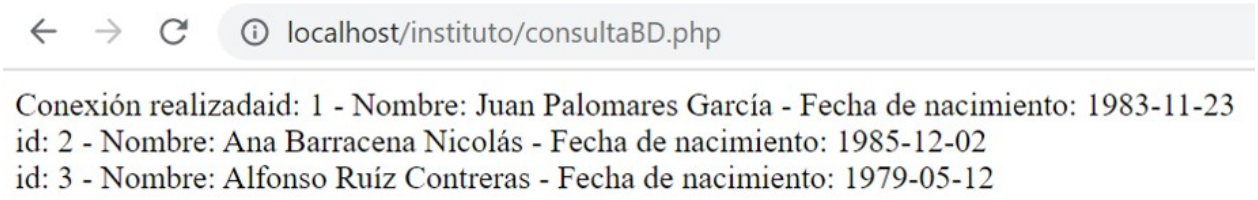
Para ello, creamos un nuevo fichero y lo llamaremos, por ejemplo, **consultaBD.php** con el siguiente código:

```
<?php
      require("conexionBD.php");

      $sql = "SELECT * FROM Alumnos";
      $result = mysqli_query($conn, $sql);

      if (mysqli_num_rows($result) > 0) {
          // datos de cada fila
          while($row = mysqli_fetch_assoc($result)) {
              echo "id: " . $row["id"]. " - Nombre: " . $row["nombre"]. "
"    .    $row["apellidos"].    "    -    Fecha    de    nacimiento:    "    .
$row["fecha_nacimiento"] . "<br>";
          }
      } else {
          echo "No hay resultados.";
      }
?>
```

➔ La consulta que hacemos aquí es muy básica: **pedimos a la base de datos que nos devuelva todos los valores de la tabla Alumnos**. Lo más interesante del código es como vamos cogiendo los datos fila a fila. Lo hacemos con `$row = mysqli_fetch_assoc($result)` para transformar las distintas filas de datos en un array asociativo.

Fíjate que siempre necesitamos el archivo **conexionBD.php** donde escribimos el código para conectarnos a la base de datos.

➔ Cuando **ejecutemos el script** veremos algo parecido a la siguiente imagen en función de los datos introducidos a la base de datos:




*Resultado de la consulta de todos los registros de la tabla alumnos.*

## Cuestionario

**Lee el enunciado e indica la opción correcta**:

¿Qué es PDO?

**a**. Printable Data Object.

**b**. PHP Data Objects.

**c**. PHP Document Objects.

**Lee el enunciado e indica la opción correcta**:

¿Qué método de PHP utilizamos para conectarnos a la base de datos?

**a**. mysqli_query().

**b**. mysqli_exec().

**c**. mysqli_connect().

**Lee el enunciado e indica la opción correcta**:

¿Qué método de PHP utilizamos para ejecutar sentencias de SQL?

**a**. mysqli_query().

**b**. mysqli_connect().

**c**. mysqli_exec().

# 5. VISUALIZACIÓN DE LA INFORMACIÓN EN PÁGINAS WEB

En el apartado anterior ya hemos sido capaces de mostrar los datos extraídos de una base de datos sobre una página web. Aunque funcional, el código anterior presenta algunos problemas:

- Aparece la frase "Conexión realizada" del archivo **conexionBD.php**
- Los datos aparecen sin formato.
- Quizás algunos datos no deberían aparecer, como por ejemplo el id.

Vamos a solucionar estos inconvenientes acometiendo el **paso 6**, el cual, a su vez, fragmentaremos en un **proceso de tres partes**:

➔ **1. Editar el archivo**:

Primero habrá que editar el archivo **conexionBD.php** para que si la conexión se ha realizado con éxito nos permita continuar sin mostrar nada por pantalla. Reescribimos el script cambiando la línea: `echo "Conexión realizada";`

por: `echo "<script>console.log('Conexión realizada');</script>";`

Así veremos si la conexión se ha realizado, pero no en pantalla, sino en la consola del navegador.

➔ **2. Reescribir consultaBD.php**:

Ahora es momento de darle un formato de tabla a los datos. Para ello crearemos una tabla con HTML y añadiremos la librería DataTables para añadirle estilo y funcionalidad. También ocultaremos el campo id. Reescribe **consultaBD.php** con el código que encontrarás en **Recursos para Ampliar con el nombre "Código paso 6".**

### ➔ 3. Resultados:

Ahora si ejecutamos el script veremos los mismos resultados, pero presentados de una forma más óptima y con las características que nos ofrece DataTables, como el filtrado, la ordenación y la paginación.


*Resultados formateados de una consulta a la base de datos.*

*Vídeo: consulta e inserción de datos*

VÍDEO

*Visualiza el siguiente vídeo en el que aprenderás cómo consultar registros y cómo insertar un nuevo registro.*

## Actividad de aprendizaje 1: conexión a base de datos

A partir del ejemplo de la tabla Alumnos visto en la teoría, crea la tabla Profesores y permite listar e insertar profesores.

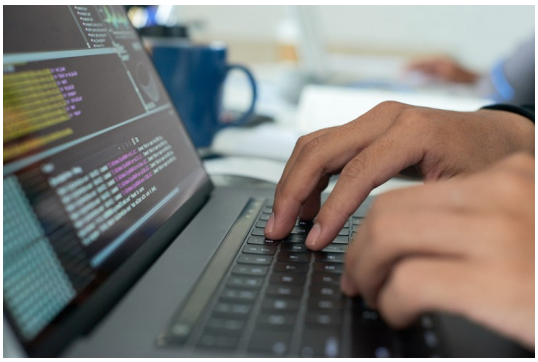
Los datos que necesitamos de cada profesor son:

- Nombre.
- Apellidos.
- Asignaturas.

En vez de pasar los parámetros vía GET como hemos visto en la teoría, crea un formulario con los datos requeridos y mándalos vía POST al servidor. Para ello busca una pareja de trabajo donde uno/a se encargará de la parte del formulario (frontend) y otro/a de la parte del script de servidor (backend).

# 6. MECANISMOS DE EDICIÓN DE LA INFORMACIÓN EN UN CLIENTE WEB

Para **editar** un alumno y cambiarle, por ejemplo, el nombre, habrá que crear un **script** que sea capaz de saber en qué alumno hay que hacer cambios y qué campos cambian.



En un entorno de producción real, esto se haría a partir de un **formulario**. Pero, centrándonos en lo estricto de este curso, recogeremos los datos vía **GET** y lanzaremos la **consulta** contra la base de datos.

*Paso 7*

Crea el archivo **edicionBD.php** y copia el siguiente script:

```php
<?php
	require("conexionBD.php");

	if (!isset($_GET['id'])) {
		exit;
	}

	$id = $_GET['id'];
	$nombre = isset($_GET['nombre']) ? ",`nombre`='" . $_GET['nombre'] . "'" : "";
	$apellidos = isset($_GET['apellidos']) ? ",`apellidos`='" . $_GET['apellidos'] . "'" : "";
	$fecha_nacimiento = isset($_GET['fecha_nacimiento']) ? ",`fecha_nacimiento`='" . $_GET['fecha_nacimiento'] . "'" : "";

	$sql = "UPDATE `alumnos` SET `id`=" . $id . $nombre . $apellidos . $fecha_nacimiento . " WHERE `id`=" . $id;

	if (mysqli_query($conn, $sql)) {
		header('Location: consultaBD.php');
	} else {
		echo "Error: " . $sql . "<br>" . mysqli_error($conn);
	}

?>
```


*Si, por ejemplo, queremos cambiar la fecha de nacimiento del usuario con id 1, deberíamos escribir en la barra de direcciones:* `http://localhost/instituto/edicionBD.php?id=1&fecha_nacimiento=1983-11-24`

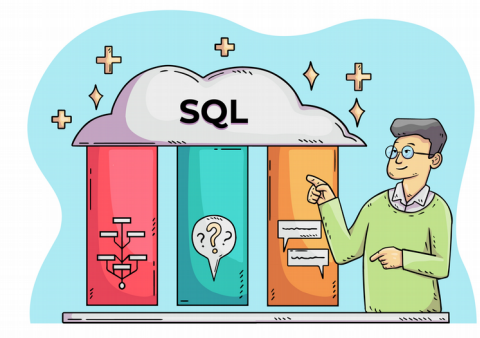
# 7. EJECUCIÓN DE SENTENCIAS SQL. SELECCIÓN, INSERCIÓN, MODIFICACIÓN Y BORRADO DE REGISTROS

En los puntos anteriores ya hemos visto la conexión entre la **sintaxis de PHP** para generar lógica, y la **sintaxis de MySQL** para realizar peticiones a la base de datos.

RECUERDA

***PHP** es un lenguaje que está dentro del paradigma imperativo y **SQL** forma parte de un paradigma declarativo.*

Ahora, vamos a hacer un repaso y a unir todas las **sentencias SQL** que necesitamos para crear, consultar, editar y modificar registros en una base de datos.




➔ **Consulta**:

- ✔ Sentencia SQL: `SELECT * FROM [tabla]`
- ✔ Ejemplo PHP: `SELECT * FROM Alumnos`

➔ **Inserción**:

- ✔ Sentencia SQL: `INSERT INTO `[tabla]`(`[clave]`) VALUES ('[valor]')`
- ✔ Ejemplo PHP: `INSERT INTO `Alumnos` (`nombre`, `apellidos`, `fecha_nacimiento`) VALUES ('" . $_GET['nombre'] . "', '" . $_GET['apellidos'] . "', '". $_GET['fecha_nacimiento'] . "')`

➔ **Edición**:

- ✔ Sentencia SQL: `UPDATE `[tabla]` SET `[clave]`='[valor]' WHERE 1`
- ✔ Ejemplo PHP: `UPDATE `Alumnos` SET `id`=" . $id . $nombre . $apellidos . $fecha_nacimiento . " WHERE `id`=" . $id;`

➔ **Eliminación**:

- ✔ Sentencia SQL: `DELETE FROM `[tabla]` WHERE 0`

✔ Ejemplo PHP: `DELETE FROM `Alumnos` WHERE `id`=$id;`


*Para ejecutar cualquiera de las sentencias anteriores sobre la base de datos habrá que utilizar la función mysqli_query() de la siguiente forma:* `mysqli_query([conexión], [sentencia]);`.

# 8. TRANSACCIONES

DESTACADO

*Las **transacciones de bases de datos** permiten agrupar sentencias (por ejemplo, SQL) en bloques, que van a ser ejecutados simultáneamente de tal forma que se pueda evaluar si alguna de las sentencias ha fallado, y de ser así, poder deshacer los cambios en el momento sin alterar de forma alguna la base de datos.*

A continuación, veremos los **objetivos de las transacciones**, las **propiedades básicas** y los **comandos básicos de control**, así como un ejemplo de una transacción:

- Las transacciones tienen dos **objetivos principales**:
  - **Proporcionar secuencias de trabajo fiables** que permitan poder recuperarse fácilmente ante errores y mantener una base de datos consistente incluso frente a fallos del sistema.
  - **Proporcionar aislamiento** entre programas accediendo a la vez a la base de datos.

- Las transacciones siguen **cuatro propiedades básicas**, bajo el acrónimo ACID (Atomicity, Consistency, Isolation, Durability):
  - **Atomicidad**: aseguran que todas las operaciones dentro de la secuencia de trabajo se completen satisfactoriamente. Si no es así, la transacción se abandona en el punto del error y las operaciones previas retroceden a su estado inicial.
  - **Consistencia**: aseguran que la base de datos cambie estados en una transacción exitosa.

- ✔ **Aislamiento**: permiten que las operaciones sean aisladas y transparentes unas de otras.
- ✔ **Durabilidad**: aseguran que el resultado o efecto de una transacción completada permanezca en caso de error del sistema.

➔ Existen **tres comandos básicos** de control en las transacciones SQL:

- ✔ **COMMIT**. Para guardar los cambios.
- ✔ **ROLLBACK**. Para abandonar la transacción y deshacer los cambios que se hubieran hecho en la transacción.
- ✔ **SAVEPOINT**. Crea checkpoints, puntos concretos en la transacción donde poder deshacer la transacción hasta esos puntos.


*Los comandos de control de transacciones se usan sólo con **INSERT, DELETE y UPDATE**. No pueden utilizarse creando tablas o vaciándolas porque las operaciones se guardan automáticamente en la base de datos.*

➔ Vamos a ver un **ejemplo de una transacción** donde agregaremos un nuevo usuario y editaremos uno existente en la misma transacción:

```
$conn=mysqli_connect("localhost","root","","Instituto");


if (mysqli_connect_errno()) {
  echo "Error al conectar a la base de datos: " . mysqli_connect_error();
  exit;
}
mysqli_autocommit($con,FALSE);
mysqli_query($conn,"INSERT INTO Alumnos (nombre,apellidos,fecha_nacimiento) VALUES (Pedro,Caro,35)");
mysqli_query($conn,"UPDATE Alumnos SET nombre = 'María' WHERE 'id' = 3 ");


if (!mysqli_commit($conn)) {
  echo "Error en la transacción";
  exit();
}
```

Al cambiar el autocommit a «false» estamos indicando que queremos enviar los cambios de forma manual que es justo lo que hacemos al final con mysqli_commit().

*Vídeo: transacciones MySQLi*


*Visualiza el siguiente vídeo en el que aprenderás a generar transacciones, qué son y cómo implementarlas.*
*https://www.youtube.com/embed/oeKfWyL2m58*

## Cuestionario

**Lee el enunciado e indica la opción correcta**:

El método mysqli_fetch_assoc:

**a**. Realiza una consulta sobre la base de datos.

**b**. Crea una conexión con la base de datos.

**c**. Obtiene una fila del resultado de una consulta como array asociativo.

**Lee el enunciado e indica la opción correcta**:

La sentencia SQL para actualizar datos de una tabla es:

**a**. SELECT.

**b**. UPDATE.

**c**. INSERT.

**Lee el enunciado e indica la opción correcta**:

¿Cuál de las siguientes propiedades no pertenece a las transacciones?

**a**. Atomicidad.

**b**. Escalabilidad.

**c**. Consistencia.

# 9. SERIALIZACIÓN

DESTACADO

*La **serialización** es el mecanismo de convertir un objeto complejo a un tipo de datos más simple y más fácil de leer para que pueda ser transmitido o guardado.*

Cada desarrollador puede **serializar los objetos** al tipo de datos que prefiera, siendo **XML** y **JSON** los más habituales.

➔ Imaginemos que tenemos el siguiente **array**:

```
<?php
$data = array("Red", "Green", "Blue");
echo $data;
?>
```

Lo que se muestra por pantalla es sencillamente "Array" porque el método "echo" no es capaz de descifrar este tipo de datos complejo.

➔ Si quisiéramos **mandar estos datos a través de un formulario o guardarlos** en una base de datos deberíamos serializarlos de la siguiente forma:

```
<?php
$data = array("Red", "Green", "Blue");
$data_serialized = serialize($data);
echo $data_serialized;
?>
```

Ahora por pantalla ya vemos el contenido de este array de la siguiente forma: `a:3:{i:0;s:3:"Red";i:1;s:5:"Green";i:2;s:4:"Blue";}`

➔ Si ahora quisiéramos **devolver estos datos a su estado inicial**, es decir, devolverlo a su formato de array deberíamos utilizar el método unserialize():

```
<?php
$data = array("Red", "Green", "Blue");
$data_serialized = serialize($data);
echo $data_serialized;
echo unserialize($data_serialized);
?>
```


*Actividad de aprendizaje 2: conexión a base de datos II*

*Siguiendo la actividad 1, añade a tu aplicación la funcionalidad de editar y eliminar profesores.*

*Añade tus propios estilos en la presentación de los resultados por pantalla y comparte tu diseño con tus compañeros.*

# 10. UTILIZACIÓN DE OTROS ORÍGENES DE DATOS

Aunque hasta ahora hemos estado utilizando **MySQLi** de forma procedimental, hemos explicado en el punto 2 de esta misma unidad didáctica que hay otras formas de conectarnos a una base de datos. Una de ellas es **PDO**.

DESTACADO

***PDO*** *son las siglas de* ***PHP Data Objects****. Tiene una estrategia parecida a MySQLi orientado a objetos, pero tiene la ventaja de trabajar con hasta 12 motores de base de datos distintos.*

Como ya sabemos trabajar con **MySQLi** de forma procedimental, vamos a ver un par de ejemplos de cómo crear una conexión con **MySQLi** orientado a objetos y con PDO:

➔ **MySQLi Orientado a Objetos**:

```
<?php
$servername = "localhost";
$username = "root";
$password = "";

$conn = new mysqli($servername, $username, $password);

if ($conn->connect_error) {
  die("Error en la conexión: " . $conn->connect_error);
}
echo "Se ha establecido la conexión";
?>
```

➔ **PDO**:

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo " Se ha establecido la conexión ";
} catch(PDOException $e) {
  echo " Error en la conexión: " . $e->getMessage();
}
?>
```

- La gran **diferencia** que vemos aquí es la línea: `$conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);` ya que si en vez de querernos conectar a una base de datos MySQL lo quisiéramos hacer sobre una PostrgreSQL, por ejemplo, sólo deberíamos cambiar `mysql:host` por `pgsql:host`.

# 11. DOCUMENTACIÓN DE LAS APLICACIONES

Cada empresa, grupo de trabajo, o incluso un programador autónomo, tiene un **manual de estilos de referencia** a la hora de programar. Esto es así por varios motivos: coherencia personal, eficiencia en el trabajo y, sobre todo, hacer más inteligible el código fuente.

**No existe un manual de estilos mejor que otro**. Cada equipo de trabajo pacta alguno. Sí es cierto que a menudo nos fijamos en manuales de estilos de grandes corporaciones a modo de referencia. Lo importante es que todo el equipo de desarrolladores trabaje de la **misma forma**. Esto hará más fácil compartir código, pedir ayuda, depurar errores, etc.


*En manuales y guías de buenas prácticas encontrarás muchas, y para todos los lenguajes de programación. Por ejemplo, si trabajas con PHP puedes fijarte con las **PHP Standard Recommendations** de la entidad PHP Framework Interop Group.*

Si prefieres montarte tú mismo los estilos, lo más importante es que seas **coherente** contigo mismo, es decir, si decides declarar variables utilizando camel case, hazlo siempre así y no cambies arbitrariamente a declararlas con delimitadores: tableroDeJuego o `tablero_de_juego`.

Aquí tienes algunos **consejos**:

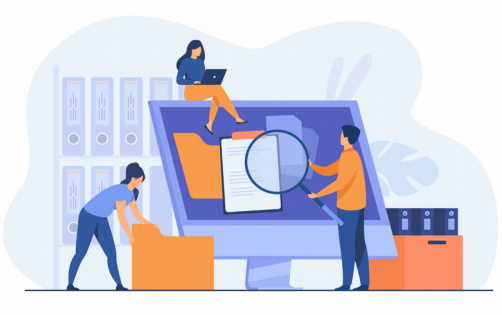
- **Nombres de variables y métodos con significado**: procura evitar a los clásicos `x` ó `y` como nombres de variables. Incluso si es un contador aislado procura llamarlo `cont`, o de la forma más descriptiva posible.

- **Sangrado (identation)**: en algunos lenguajes es obligatorio (Python), pero en la mayoría no. Sin embargo, tener el código bien sangrado lo hace más legible.
- **Espaciado**: la mayoría de lenguajes ignoran los espacios, pero al igual que con el sangrado, es una buena manera de visualizar el código. Procura colocar espacios entre los parámetros de las funciones, entre operadores, etc.
- **Métodos coherentes**: es mejor tener pequeñas funciones encargadas de una sola tarea, que una función que haga muchas cosas.
- **Claves**: utiliza siempre claves, incluso para blogs de código que sólo tienen una sentencia.
- **Longitudes y saltos de línea**: escribe sólo una sentencia por línea, evitando que las líneas sean infinitamente largas, aunque haya IDEs y editores de texto que surtan efecto wrapping (envolver el código).
- **Paréntesis**: utiliza paréntesis en expresiones que impliquen a diferentes operadores, sobre todo para señalar la prioridad. Aunque a ti te parezca obvio, quizás no todo el mundo lo ve con la misma facilidad.

# 12. ALMACENES DE INFORMACIÓN HETEROGÉNEOS

Imaginemos que en nuestra tabla Alumnos queremos guardar las notas de estos alumnos. Esto, en un modelo relacional al uso, se debería resolver creando la tabla "notas" y quizás la tabla "asignaturas".

Pero si queremos aprovechar la estructura ya creada nos puede parecer interesante **mezclar un modelo relacional con un modelo no relacional**, es decir, crear una base de datos heterogénea.



- Si queremos **crear un campo donde guardar información no relacional** podemos crearlo y asignarle el tipo JSON con la siguiente instrucción de SQL:

```
ALTER TABLE `alumnos` ADD `notas` JSON NOT NULL AFTER `fecha_nacimiento`;
```

- Ahora dentro de este campo **agregamos un objeto JSON** con notas de distintas asignaturas:

```
[{
            "modulo": "M1",
            "nota": "10"
      },
      {
            "modulo": "M2",
            "nota": "7"
      },
      {
            "modulo": "M3",
            "nota": "8"
      }
]
```

➔ Para **probar que funciona** cambiaremos un poco el código del script **consultaBD.php** agregándole un foreach() para que vaya recorriendo aquello que encuentre dentro del campo "notas" de la tabla Alumnos, previa conversión de los datos serializados JSON al objeto Array con las notas.


*Este código lo encontrarás en recursos para ampliar con el nombre "código – almacenes de información heterogéneos"*

*Vídeo: modificación de datos y almacenes de información heterogéneos*


*Visualiza el siguiente vídeo que tratará sobre la modificación de datos y almacenes de información heterogéneos.*

*Actividad de aprendizaje 3: conexión a base de datos III*


*Cambia (o añade) un nuevo archivo de conexión a la base de datos de las actividades anteriores, pero esta vez hazlo con PDO.*

*Documenta y escribe tu código siguiendo las buenas prácticas vistas en la teoría.*

## Cuestionario

**Lee el enunciado e indica la opción correcta**:

¿Qué es la serialización?

**a**. Un mecanismo de convertir un objeto complejo a un tipo de datos más simple.

**b**. Ordenar objetos uno después de otro dentro de un array.

**c**. Una forma de conectar componentes.

**Lee el enunciado e indica la opción correcta**:

El método de PHP para serializar un objeto es:

**a**. Serial().

**b**. Serialize().

**c**. Mysql_serialize().

**Lee el enunciado e indica la opción correcta**:

La ventaja de PDO sobre MySQLi es:

**a**. Que es más rápido.

**b**. Que puede trabajar con distintos motores de base de datos.

**c**. Que es más fácil.

## *RESUMEN*

En esta unidad didáctica hemos aprendido **cómo conectarnos a una base de datos** e **interactuar** con ella creando nuevos registros, eliminando, consultando y modificando.

Con ello, y sin darnos mucha cuenta hemos creado un **CRUD**. En informática, CRUD es el acrónimo de "**Crear, Leer, Actualizar y Borrar**" (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

También hemos visto que hay datos que no se pueden guardar "tal cual" en una base de datos y primero hay que **serializarlos** para luego **deserializarlos**.

Finalmente hemos trabajado con **almacenes de datos heterogéneos** donde conviven tecnologías relacionales con no relacionales.

## *RECURSOS PARA AMPLIAR*

### PÁGINAS WEB

- Mayúsculas y minúsculas en MySQL: *https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html* [Consulta noviembre 2022].
- Tutorial de MySQL de w3schools: *https://www.w3schools.com/mysql/default.asp* [Consulta noviembre 2022].

### TEXTOS ELECTRÓNICOS

- Código - Almacenes de información heterogéneos: *http://imagenes.contenidosteleformacion.es/ampliar/ense2021/unidad6/codigo_alm_heterogeneos.pdf* [Consulta noviembre 2022].
- Código – Paso 6: *http://imagenes.contenidosteleformacion.es/ampliar/ense2021/unidad6/codigo_paso6.pdf* [Consulta noviembre 2022].

# BIBLIOGRAFÍA

## PÁGINAS WEB


- Documentación de MySQL: *https://dev.mysql.com/doc/* [Consulta noviembre 2022].
- Documentación de phpMyAdmin: *https://www.phpmyadmin.net/docs/* [Consulta noviembre 2022].
- Referencia oficial de PHP: *https://www.php.net/* [Consulta noviembre 2022].


## GLOSARIO

- **JSON**: siglas de JavaScript Object Notation. JSON es un formato de texto sencillo destinado a enviar y/o almacenar estructuras de datos.
- **PDO**: siglas de PHP Data Objects. Es un mecanismo de PHP para crear una capa de abstracción para poder guardar datos complejos en una base de datos.
- **SQL**: siglas de Structured Query Language. SQL es un lenguaje de dominio específico, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.
- **XML**: siglas de eXtensible Markup Language. XML es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.