



INSTITUT



## Unión Europea

Fondo Social Europeo

*El FSE invierte en tu futuro*

**PHP**

Acceso a Datos en PHP

Pepe



**GENERALITAT  
VALENCIANA**

Conselleria d'Educació,  
Investigació, Cultura i Esport



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE EDUCACIÓN, CULTURA  
Y DEPORTE

INSTITUT

## Continguts

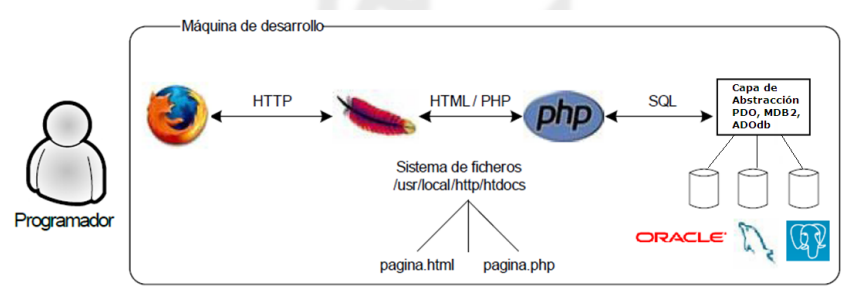
|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Pròleg</b>                          | <b>3</b>  |
| <b>2</b> | <b>Preparación de la base de datos</b> | <b>3</b>  |
| <b>3</b> | <b>Configuración de la conexión</b>    | <b>4</b>  |
| 3.1      | PHP Data Objects :: PDO . . . . .      | 5         |
| <b>4</b> | <b>Consultas Preparadas</b>            | <b>6</b>  |
| 4.1      | Insert(CRUD) . . . . .                 | 6         |
| 4.2      | Parametrizar operaciones . . . . .     | 6         |
| 4.3      | READ (leer) . . . . .                  | 8         |
| 4.4      | UPDATE (Actualizar) . . . . .          | 9         |
| 4.5      | DELETE (Borrar) . . . . .              | 9         |
| <b>5</b> | <b>Liberar las conexiones</b>          | <b>9</b>  |
| <b>6</b> | <b>SQL Injection</b>                   | <b>10</b> |
| <b>7</b> | <b>TRANSACCIONES</b>                   | <b>14</b> |
| <b>8</b> | <b>Ejercicio</b>                       | <b>14</b> |

## 1 Pròleg

En esta unidad vamos a aprender a acceder a datos que se encuentran en un servidor; recuperando, editando y creando dichos datos a través de una base de datos.

A través de las distintas capas o niveles, de las cuales 2 de ellas ya conocemos (Apache, PHP) y MySQL la que vamos a estudiar en este tema.

En la siguiente figura queda más claro donde se colocaría la capa de acceso de abstracción o capa de acceso a datos.



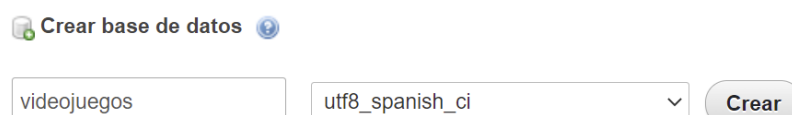
(a) Gestión

## 2 Preparación de la base de datos

Antes de conectar con una base de datos, evidentemente tenemos que tenerla creada y lista. Si utilizamos XAMPP, este paso puede hacerse fácilmente a través de la herramienta phpMyAdmin que ya viene instalada. Para usarla, accedemos a su URL predeterminada, que suele ser <http://localhost/phpmyadmin>.



(b) phpmyadmin



(c) phpmyadmin

| Nombre | Tipo    | Longitud/Valores | Predeterminado | Cotejamiento | Atributos | Nulo                     | Índice  | A.                                  |
|--------|---------|------------------|----------------|--------------|-----------|--------------------------|---------|-------------------------------------|
| id     | INT     |                  | Ninguno        |              |           | <input type="checkbox"/> | PRIMARY | <input checked="" type="checkbox"/> |
| titulo | VARCHAR | 100              | Ninguno        |              |           | <input type="checkbox"/> | --      | <input type="checkbox"/>            |
| genero | VARCHAR | 50               | Ninguno        |              |           | <input type="checkbox"/> | --      | <input type="checkbox"/>            |
| precio | REAL    |                  | Ninguno        |              |           | <input type="checkbox"/> | --      | <input type="checkbox"/>            |

(d) phpmyadmin

Con esto, ya tendremos la tabla videojuegos creada en la base de datos. Haciendo clic en ella desde el panel principal podremos consultar la información que haya guardada en cada momento.

Además, desde las opciones del menú superior Importar y Exportar podemos incorporar nuevas bases de datos (previamente exportadas) o exportar el contenido de las existentes para hacer copias de seguridad o llevarlas a otro servidor.

### 3 Configuración de la conexión

A la hora de conectar con cualquier base de datos, tenemos que tener en cuenta cuatro o cinco parámetros clave:

- **Dirección del servidor de bases de datos.** Típicamente suele estar alojado en la misma máquina donde tenemos el servidor web Apache, así que esta dirección suele ser localhost.
- **Puerto de conexión con el servidor.** Normalmente cada servidor queda escuchando por su puerto por defecto y no es necesario configurarlo. Así, el puerto por defecto de MySQL es el 3306, por ejemplo.
- **Nombre de la base de datos a la que queremos conectar.**
- **Login y password del usuario con el que queremos conectar.** En el caso de XAMPP, por defecto se crea un usuario root con contraseña vacía.

Teniendo todo esto en cuenta, el siguiente código nos va a permitir conectar con una base de datos como la que hemos creado en el paso anterior:

```
$host = "localhost";
$nombreBD = "videojuegos";
$usuario = "root";
$password = "";

$pdo = new PDO("mysql:host=$host;dbname=$nombreBD;charset=utf8",$usuario,
    $password);
```

### 3.1 PHP Data Objects :: PDO

De la misma manera podríamos hacerlo con **mysqli**, PHP Data Objects (o PDO) es un driver de PHP que se utiliza para trabajar bajo una interfaz de objetos con la base de datos. A día de hoy es lo que más se utiliza para manejar información desde una base de datos, ya sea relacional o no relacional. El objeto *PDO* anterior se construye usando tres parámetros:

- URL de conexión, donde indicamos el tipo de base de datos que vamos a usar (MySQL, en este caso), dirección del servidor, nombre de la base de datos y codificación. *Notar que simplemente cambiando el tipo de base de datos (PostgreSQL, Oracle...) esta misma línea nos va a servir para conectar con distintos SGBD.*
- Usuario y contraseña de acceso.



Estas líneas de conexión van a resultar muy frecuentes en nuestras aplicaciones si accedemos a la base de datos desde distintas páginas PHP, por lo que convendría definir las en un archivo `.inc` e incluirlo en las páginas que necesiten esa conexión.

Además, con PDO podemos usar las excepciones con **try catch** para gestionar los errores que se produzcan en nuestra aplicación, para ello, como hacíamos antes, debemos *encapsular* el código entre bloques `try / catch`.

```
<?php
$dsn = 'mysql:dbname=prueba;host=127.0.0.1';
$usuario = 'usuario';
$contraseña = 'contraseña';

try {
    $pdo = new PDO($dsn, $usuario, $contraseña);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Falló la conexión: ' . $e->getMessage();
}

?>
```

En primer lugar, creamos la conexión con la base de datos a través del constructor PDO pasándole la información de la base de datos. En segundo lugar, establecemos los parámetros para manejar las excepciones, en este caso hemos utilizado:

- **PDO::ATTR\_ERRMODE** indicándole a PHP que queremos un reporte de errores.
- **PDO::ERRMODE\_EXCEPTION** con este atributo obligamos a que lance excepciones, además de ser la opción más humana y legible que hay a la hora de controlar errores.

Cualquier error que se lance a través de PDO, el sistema lanzará una **PDOException**.

## 4 Consultas Preparadas

### 4.1 Insert(CRUD)

Para ejecutar instrucciones SQL, seguiremos dos pasos:

1. Preparamos la instrucción SQL a ejecutar (*SELECT*, *INSERT*, *UPDATE*, *DELETE*). Utilizaremos la instrucción `prepare` para ello.
2. La ejecutamos, indicando si es necesario algunos parámetros variables en la operación (valores para algunas condiciones o campos). Emplearemos la instrucción `execute` para esta ejecución.

Así lanzaríamos una instrucción *INSERT* para insertar datos fijos en nuestra tabla videojuegos del ejemplo anterior, una vez obtenida la conexión en el objeto `$pdo` anterior.

```
$insercion = $pdo->prepare("INSERT INTO videojuegos(titulo, genero, precio) VALUES('Fifa 2020', 'Deportes', 40.95)");  
$insercion->execute();
```

```
$sql="INSERT INTO videojuegos(titulo, genero, precio) VALUES('Fifa 2020', 'Deportes', 40.95)";  
$insercion = $pdo->prepare($sql);  
$insercion->execute();
```



Las sentencias con PDO evitan la inyección de SQL (*SQL Injection*) y mejoran el rendimiento de nuestras aplicaciones o páginas web.

### 4.2 Parametrizar operaciones

Lo normal es que no hagamos operaciones con todos los datos fijos, sino que parte de la query dependa de ciertos parámetros externos (datos que nos llegan de un formulario, por ejemplo). La función `prepare()` prepara la consulta como sentencia predefinida. Se pueden utilizar tanto el signo de interrogación (?) como dos puntos seguidos del nombre de la variable de la plantilla (**:variable**) como marcadores de las plantillas.

- Versión con interrogantes:

```
$titulo="WWII";
$genero="Bélico";
$precio=12.2;

$consulta = "INSERT INTO videojuegos(titulo, genero, precio)" .
            " VALUES(?, ?, ?)";

$insertion = $pdo->prepare($consulta);

$insertion->bindParam (1, $titulo, PDO::PARAM_STR);
$insertion->bindParam (2, $genero, PDO::PARAM_STR);
$insertion->bindParam (3, $precio);

$insertion->execute();
```

- Versión marcadores:

```
$consulta = "INSERT INTO videojuegos(titulo, genero, precio)" .
            " VALUES(:titulo, :genero, :precio)";
$insertion = $pdo->prepare($consulta);
$insertion->bindParam (':titulo', $titulo, PDO::PARAM_STR);
$insertion->bindParam (':genero', $genero, PDO::PARAM_STR);
...
$insertion->execute();
```

Mediante **bindParam()** emparejamos la variable con el valor de la plantilla. Es mucho más cómodo porque no tenemos que seguir el orden establecido por los signos de interrogación.

Datos supuestamente recogidos de un envío de formulario

```
$insertion = $pdo->prepare("INSERT INTO videojuegos(titulo, genero, precio)
    )" .
    " VALUES(:titulo, :genero, :precio)");
$insertion->bindParam(':titulo', $_REQUEST['titulo']);
$insertion->bindParam(':genero', $_REQUEST['genero']);
$insertion->bindParam(':precio', $_REQUEST['precio']);
$insertion->execute();
```

La función **bindParam()** admite otro parámetro para establecer el tipo del dato a insertar. Por defecto se establece a PARAM\_STR (STRING) [Tipos de datos](#)

Más información en el manual: [bindParam](#)

Por último, el método **execute()** ejecuta la sentencia preparada. Esta función admite un parámetro optativo, más concretamente un array de variables para emparejar con los marcadores.

Veamos un ejemplo:

```
$insertion = $pdo->prepare("INSERT INTO videojuegos(titulo, genero, precio)
    )" .
```

```
" VALUES(:titulo, :genero, :precio)");  
$videojuegos = array ( "Done to Zen", "Belica", 23.6);  
$insercion->execute($videojuegos);
```

### 4.3 READ (leer)

```
$consulta = $pdo->query("SELECT * FROM videojuegos");  
$consulta->execute();  
while($registro = $consulta->fetch())  
{  
    echo $registro['titulo']."<br>";  
}
```

- Parametrizado

```
$consulta = $pdo->query("SELECT * FROM videojuegos WHERE genero=:genero");  
$consulta->bindParam(':genero', $_REQUEST['genero']);  
$consulta->execute();  
while($registro = $consulta->fetch())  
{  
    echo $registro['titulo']."<br>";  
}
```

#### 4.3.1 Consultando registros

A la hora de recuperar los resultados de una consulta, como hemos visto, bastará con invocar al método **PDOStatement::fetch** para listar las filas generadas por la consulta.

Pero debemos elegir el tipo de dato que queremos recibir entre los 3 que hay disponibles:

- **PDO::FETCH\_ASSOC**: array(asociativo) indexado cuyos keys son el **nombre de las columnas**.
- **PDO::FETCH\_NUM**: array indexado cuyos **keys son números**.
- **PDO::FETCH\_BOTH**: valor por defecto. Devuelve un array indexado cuyos keys son tanto el nombre de las columnas como números.

```
$consulta = $pdo->prepare("SELECT * FROM videojuegos");  
$consulta -> setFetchMode(PDO::FETCH_ASSOC);  
$consulta->execute();  
while($registro = $consulta->fetch())  
{  
    echo $registro['titulo']."<br>";  
}
```



Pero si lo que queremos es leer datos con forma de objeto utilizando `PDO::FETCH_OBJ`, debemos crear un objeto con propiedades públicas con el mismo nombre que las columnas de la tabla que vayamos a consultar.

```
$consulta = $pdo->prepare("SELECT * FROM videojuegos");
$consulta -> setFetchMode(PDO::FETCH_OBJ);
$consulta->execute();
while($registro = $consulta->fetch())
{
    echo $registro->titulo."<br>";
}
```

#### 4.4 UPDATE (Actualizar)

Para actualizar datos:

```
try {
    $sql = "UPDATE videojuegos SET titulo = :titulo WHERE id = :id";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['titulo' => 'Asoc.', 'id' => 143]);
    echo "Usuario actualizado con éxito.";
} catch (\PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

#### 4.5 DELETE (Borrar)

Para borrar datos:

```
try {
    $sql = "DELETE FROM videojuegos WHERE id = :id";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['id' => 1]);
    echo "Usuario borrado con éxito.";
} catch (\PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

### 5 Liberar las conexiones

Una vez hemos terminado de trabajar con la base de datos (en cada página donde lo hayamos hecho), para liberar la conexión y que la pueda utilizar otro cliente que quiera acceder, debemos anular (asignar

a NULL) tanto el objeto de la conexión (\$pdo en los ejemplos anteriores) como el que hemos utilizado para realizar la operación (variables \$insercion o \$consulta en los ejemplos anteriores). Por ejemplo:

```
$consulta = NULL;  
$pdo = NULL;
```

## 6 SQL Injection

Vamos a modelar nuestra inyecciones SQL. A parte de la tabla de videojuegos, tendremos una tabala de usuarios como la siguiente:

| Id | nombre   | password | rol   |
|----|----------|----------|-------|
| 1  | pepe     | pepe     | admin |
| 2  | usuario1 | usu1     | user  |
| 3  | uuario2  | usu2     | user  |

El siguiente codigo con MySQLI

```
<?php  
  
// Datos  
$dbhostname = 'localhost';  
$dbuser = 'root';  
$dbpassword = '';  
$dbname = 'videojuegos';  
  
//Creamos la conexion  
$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);  
  
//Comprobamos si se ha hecho bien la conexion  
  
// Parametro con el cual recogemos el input  
$input= $_GET['genero'];  
var_dump($input);  
// Definimos consulta a Mariadb  
$query = "SELECT id,titulo,genero, precio FROM videojuegos WHERE genero='  
    $input'";  
  
// Lanzamos la consulta  
$results = mysqli_query($connection, $query);  
  
// Comprobamos si se ha hecho bien la consulta
```

```
if (!$results) {
    echo mysqli_error($connection);
    die();
}


echo "<table>";
echo "<tr>";
echo "  <th align='left'> ID </th>";
echo "  <th align='left'> Producto </th>";
echo "  <th align='left'> genero </th>";
echo "  <th align='left'> precio </th>";
echo "</tr>";

// Obtenemos y mostramos ls resultados de la consulta. Los resultados se
// almacenan en un array por el cual iteramos
while ($rows = mysqli_fetch_assoc($results)) {

    echo "<tr>";
    echo "<td align='left'> " . $rows['id'] . "</td>";
    echo "<td align='left'> " . $rows['titulo'] . "</td>";
    echo "<td align='left'> " . $rows['genero'] . "</td>";
    echo "<td align='left'> " . $rows['precio'] . "</td>";
    echo "</tr>";
    echo "</table>";
}

?>
```

Hagamos una primera consulta desde la URL



string(8) "Deportes"

| ID | Producto | Descripcion | precio |
|----|----------|-------------|--------|
|----|----------|-------------|--------|

|     |         |          |       |
|-----|---------|----------|-------|
| 154 | FIFA 21 | Deportes | 49.99 |
|-----|---------|----------|-------|

|     |          |          |       |
|-----|----------|----------|-------|
| 155 | NBA 2K21 | Deportes | 59.99 |
|-----|----------|----------|-------|

**Figure 1:** Consulta el genero Deportes

Podemos ver si estamos ante una inyección SQL de este tipo con el siguiente payload 1' UNION SELECT 1,2,3,4-- -. Con esta consulta pueden pasar 3 cosas:

- Que añada el dato de la segunda consulta SELECT sin dar errores.
- Que nos salte el siguiente error: The used SELECT statements have a different number of columns.

**RECORDAR: con UNION tiene que tener el mismo número de columnas**

- Que no ocurra nada.

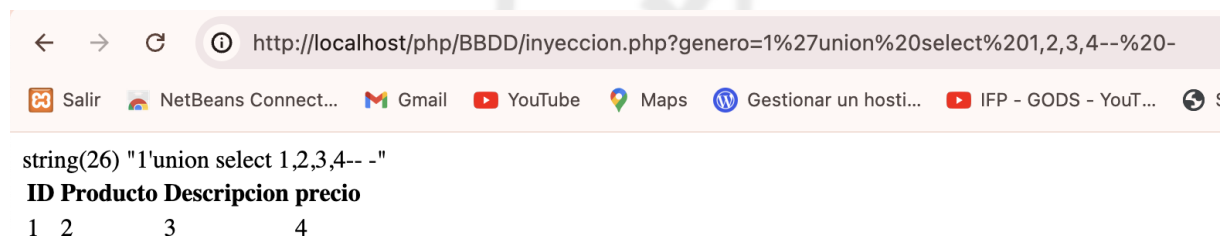
Veamos:

- En primer lugar escribimos:

 `http://localhost/php/BBDD/inyeccion.php?genero=1'union select 1,2,3,4-- -`

(a) Payload

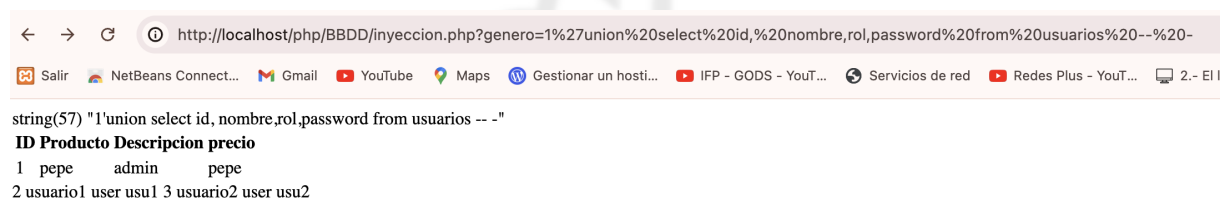
- Ejecutamos



(b) Ejecución

Vemos que hemos sido capaces de inyectar código fuera de los límites del campo del genero.

Si conoces el nombre de la tabla usuarios podemos ver el contenido inyectando el payload `1'union select id, nombre,password,rol from usuarios -- -`



(c) Usuarios

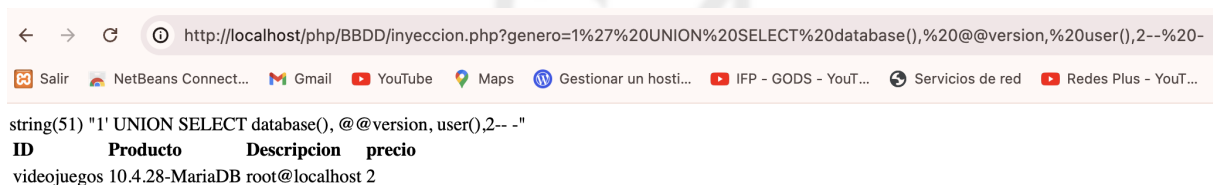
... y sacamos todo el contenido de la tabla de usuarios.

Ya sabiendo la cantidad de columnas que tiene la tabla vamos a empezar a recopilar información básica sobre la base de datos antes de empezar con la explotación.

| Información a obtener       | Software | Consulta   |
|-----------------------------|----------|------------|
| Nombre de la base de datos  | Todos    | database() |
| Versión de la base de datos | MySQL    | @@version  |

| Información a obtener         | Software   | Consulta   |
|-------------------------------|------------|------------|
|                               | Oracle     | v\$version |
|                               | PostgreSQL | version()  |
| Usuario que la está corriendo | Todos      | user()     |

Vamos a mandar el siguiente payload 1 ' UNION SELECT database(), @@version, user()-- -.



(d) Información BBDD

Ahora si intentáramos hacer lo mismo con el objeto **PDO** podréis observar que controla y elimina las inyecciones SQL.

```
$host = "localhost";
$nombreBD = "videojuegos";
$usuario = "root";
$password = "";

try {
    $pdo = new PDO("mysql:host=$host;dbname=$nombreBD;charset=utf8",
        $usuario, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

} catch (PDOException $e) {
    echo 'Falló la conexión: ' . $e->getMessage();
}

$input= $_GET['genero'];
var_dump($input);
$consulta = $pdo->prepare("SELECT titulo,genero,precio FROM videojuegos
    WHERE genero = :genero");// WHERE genero=:genero");
$consulta->bindParam(':genero', $input);
$consulta->execute();
while($registro = $consulta->fetch())
{
}
```

```
echo $registro['titulo']." ".$registro['genero']." ".$registro['precio']."  
<br>";  
}
```

## 7 TRANSACCIONES

Una transacción consiste en un conjunto de operaciones que tienen que realizarse de forma atómica. Es decir, o se realizan todas o ninguna.

Por defecto *PDO* trabaja en manera `autocommit`, así se confirma de forma automática cada sentencia que ejecuta el servidor.

Para trabajar con transacciones, PDO incorpora tres métodos:

- **beginTransaction.** Deshabilita la manera `autocommit` y empieza una nueva transacción, que finalizará cuando ejecutas uno de los dos métodos siguientes.
- **commit.** Confirma la transacción actual.
- **rollback.** Revierte los cambios llevados a cabo en la transacción actual. Una vez ejecutado un `commit` o un `rollback`, se volverá a la manera de confirmación automática.

## 8 Ejercicio

Vamos a crear un CRUD de una única tabla *task*:

- `Create`





```
CREATE TABLE task(  
  id INT(11) PRIMARY KEY AUTO_INCREMENT,  
  title VARCHAR(255) NOT NULL,  
  description TEXT,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Veamos el comportamiento con un ejemplo:

- `Read` Debemos crear un formulario para registrar las entradas:
  - Cuando pulsamos `Registrar` el contenido de los campos se deben insertar en la tabla *task* de la BBDD y deberás mostrar en un listado todos los registros incluido el nuevo, junto con la fecha de creación *created\_at*.

## EJEMPLO PHP MySQL CRUD

|                                     |  |  |
|-------------------------------------|--|--|
| <input type="text" value="Título"/> | <input type="text" value="Descripción"/> | <input type="button" value="Registrar"/> |
|-------------------------------------|--|--|

| Título          | Descripción  | Creado el:          | Acción   |
|-----------------|--|---------------------|--|
| Caperucita roja | Un lobo que se come una iaia   | 2024-10-28 10:25:49 | <br> |
| El principito   | Un piloto se encuentra perdido en el desierto del Sahara después de que su avión sufriera una avería, pero para su sorpresa, es allí donde conoce a un pequeño príncipe proveniente de otro planeta. | 2024-10-28 10:29:16 | <br> |

IES Salvador Gadea

**Figure 2:** Principal

- **Update:**
  - Seleccionaremos un registro y pulsaremos un *botón o enlace* que nos permita modificar el campo o campos en cuestión. EXCEPTO la PK.

## EJEMPLO PHP MySQL CRUD

|   |
|---|
| <input type="text" value="Caperucita roja"/>              |
| <input type="text" value="Un lobo que se come una iaia"/> |
| <input type="button" value="Update"/>                     |

IES Salvador Gadea

**Figure 3:** Actualizar

- **Delete:**

- Se seleccionará el registro y se borrará directamente cuando se pulse el *botón o enlace*.



- También sería interesante que utilizáramos una sesión y que saliese vuestro nombre como que estáis autenticados y podéis operar.
- Es importante el diseño.
- Pensar que debe de ser muy intuitivo, el usuario debe de saber donde se encuentra siempre.