

ACTIVITAT AVALUABLE AC13

Mòdul professional 7: Desenvolupament web en l'entorn servidor

UF_3: tècniques d'accés a dades

Professor: Nelson Hernández

Data límit d'entrega: Por determinar, consultar en entregues

Mètode d'entrega: Per mitjà del Clickedu de l'assignatura. Les activitats entregades més enllà de la data límit només podran obtenir una nota de 5.

Instruccions: Totes les tasques han de entregar-se en un sol document, de nom:

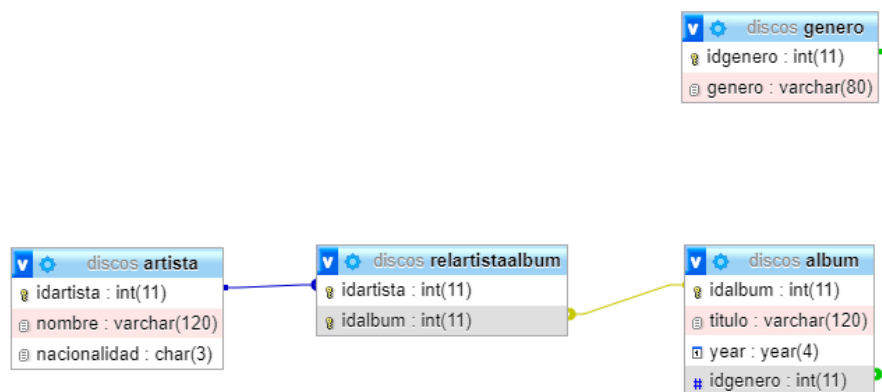
M7-UF3-AC13-Nom_Alumne

Resultats de l'aprenentatge:

RA_1: Desenvolupa aplicacions d'accés a magatzems de dades, aplicant mesures per mantenir la seguretat i la integritat de la informació.

Tasques a realitzar:
Aplicación MVC con PDO y AJAX
EJERCICIO 0: CARGA DE LA BASE DE DATOS

El primer paso será cargar la base de datos **discos.sql** que encontraréis en los recursos de la actividad, una vez cargada la base de datos podemos ver su estructura en la vista de diseñador.



Particularidades:

- Las tablas que tenemos que mantener con la plataforma son las de *artista* y *album* • La tabla *genero* únicamente la utilizaremos para confeccionar de forma dinámica la combo del formulario de la pantalla de mantenimiento de albums
- Podemos ver que entre *artista* y *album* hay una relación muchos a muchos. Tened esto en cuenta porque, cuando demos de alta o modifiquemos un album, tendremos que actualizar dos tablas: *album* y la tabla de relación *relartistaalbum* de forma que, o se actualizan las dos o no se actualiza ninguna (para evitar errores de integridad referencial). Para ello utilizaremos una transacción

EJERCICIO 1: ALTA DE ARTISTA

Nombre	Nacionalidad
Connie Corleone	ITA
Guns N' Roses	USA
Luca Brasi	ITA
Peter Clemenza	ITA
Rammstein	GER
Virgil Solozzo	ITA

1. Acciones a realizar en la Vista

Vamos a editar la vista ***artistas.html*** para añadir los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de alta.

Añadiremos dos ficheros justo antes de cerrar la etiqueta **<body>**

1. **altaartista.js**
2. **inicioartista.js**

Fichero *inicioartista.js*

En este fichero indicaremos las acciones que hay que realizar cuando se carga la vista *artista.html* y que, para el caso del alta, son las siguientes:

- Activar el listener para detectar la pulsación del botón de alta de artista

```
document.querySelector('#alta').onclick = altaArtista
```

Fichero *altaartista.js*

En este fichero definiremos la función *altaArtista* que se encargará de realizar la petición ajax al controlador del servidor para realizar el alta:

- Recuperar los datos del formulario. Ejemplo para nombre:

```
let nombre = document.querySelector('#nombre').value.trim()
```

- Informar el controlador donde vamos a realizar la petición. Ejemplo:

```
let servicio = 'servicios/controllers/artistacontroller.php'
```

- Confeccionar el objeto *FormData* de la petición. Ejemplo:

```
let datos = new FormData();
```

```
datos.append('peticion', 'A') → tipo de petición (A = alta) datos.append('nombre', nombre); →  
nombre del artista
```

```
datos.append('nacionalidad', nacionalidad); → nacionalidad artista
```

- Confeccionar el objeto con los parámetros de la petición

```
let parametros = {  
  method: 'post',  
  body: datos  
}
```

- Realizar la llamada ajax utilizando *fetch()*. Para la operativa de alta el servidor nos podría devolver un texto pero es preferible que nos devuelva un *json* con la siguiente estructura:

[*'codigo'* => 99, *'datos'* => *'datos o mensaje de respuesta'*, *'error'* => *'mensaje de error si lo hay'*] Que recogeríamos en la respuesta ajax de la siguiente forma:

```
let codigo = mensaje.codigo;
```

```
let datos = mensaje.datos;
```

```
let error = mensaje.error;
```

- Si el código de respuesta es '00':

- Reiniciamos el formulario

```
document.querySelector('#formulario').reset();
```

- Desactivamos los botones de baja y modificación. Ejemplo para baja:

```
document.querySelector('#modificar').setAttribute('disabled', true)
```

- Y, puesto que el alta que acabamos de efectuar, debe reflejarse en la tabla de consulta (que veremos después) tendremos que realizar una llamada a la función javascript que se encargará de confeccionarla

```
consultaArtistas()
```

NOTA: Recordad que en las peticiones asíncronas no se actualiza la página por lo que tenemos que actualizar la tabla mediante una nueva petición al servidor

2. Acciones a realizar en el Controlador

Vamos a confeccionar el controlador *artistacontroller.php* para añadir la operativa de alta de artista.

NOTA: Este controlador se encargará de manejar las peticiones de alta, baja, consulta y modificación que procedan de la vista y trasladará los datos al modelo para que sea este quien realice las acciones contra la base de datos. De momento veremos solo la operativa de alta

Las acciones que hay que realizar son las siguientes:

- Incorporar el fichero que utilizaremos como modelo

```
require '../models/artistamodel.php'; //recordad que se encuentra en la carpeta models
```

- Recuperar el tipo de petición que nos llega de la vista (recordad que hemos enviado un atributo 'peticion' con el valor 'A' de alta)

```
if (!$peticion = filter_input(INPUT_POST, 'peticion')) {
    throw new Exception("Petición obligatoria", 10);
}
```

- Puesto que el modelo estará confeccionado íntegramente con Orientación a Objetos y, por tanto, será una clase php tenemos que instanciar un objeto de este modelo:

```
$artista = new ArtistaModel();
```

NOTA: *\$artista* es un objeto de la clase *ArtistaModel* que confeccionaremos más adelante

- Evaluamos la petición recibida (por ejemplo con *switch(\$peticion)*) y, en caso que sea un alta:

- Recuperamos el resto de datos para el alta. Ejemplo para nombre:

```
$nombre = $_POST['nombre']; // o con filter_input()
```

- Llamar al método del modelo que se encargará del alta del artista:

```
$respuesta = $artista->alta($nombre, $nacionalidad)
```

NOTA 1 : EL modelo nos enviará una respuesta que recogeremos en la variable `$respuesta`.

NOTA 2: ¿Qué pasa si en el modelo hay alguna incidencia que genere una excepción?. Pues que la recogeremos en el controlador dentro del `catch` que deberíamos tener al final del bloque `try` en donde estamos codificando la operativa que estamos viendo en este apartado.

En el `catch` confeccionamos la respuesta para error:

```
$respuesta = array('codigo'=>$e->getCode(), 'error'=>$e->getMessage())
```

NOTA 3: Para empezar a adaptarnos a la forma de trabajar de los Frameworks, es deseable que al modelo enviemos los datos en forma de array asociativo en vez de datos individuales. Para ello:

```
$datos = compact('nombre', 'nacionalidad'); //creamos un array asociativo con los datos del alta $respuesta =
```

```
$artista->alta($datos) //enviamos al modelo el array asociativo
```

- Si tuviéremos que realizar alguna operativa adicional para el alta y que no sea responsabilidad del modelo (por ejemplo escribir un fichero, enviar un correo, etc.) la codificaríamos a continuación de la llamada al modelo y antes de confeccionar la respuesta a la vista. (no es el caso de esta actividad)

- Confeccionamos la respuesta a la vista (ya hemos decidido que va a ser siempre un `json`):

```
echo json_encode($respuesta);
```

3. Acciones a realizar en el Modelo

Vamos a confeccionar el modelo ***artista_model.php*** para añadir la operativa de alta de artista.

NOTA 1: Este modelo será una clase php con un método para cada una de las operativas de alta, baja, consulta y modificación que procedan de peticiones del controlador. Una vez se ha accedido a la base de datos trasladará el resultado de nuevo al controlador para que sea éste quien los traslade a la vista.

Como podéis observar todas las operativas del backend pasan por el controlador. Este gestiona las peticiones de la vista y las envía al modelo y, a la vez, recoge las respuestas del modelo y las envía a la vista. EL modelo jamás enlaza directamente con la vista ni la vista con el modelo.

NOTA 2: En este ejemplo se va a utilizar la librería PDO

Las acciones que hay que realizar son las siguientes:

- Incorporar el fichero de conexión a la base de datos:

```
require 'database.php';
```

NOTA: Para realizar el ejercicio se debe utilizar la librería PDO (PHP Data Objects) - como recurso de consulta podemos utilizar [w3Schools](http://w3schools.com) o la ayuda del profesor

```

class Database {
protected $conexion;
public function __construct() {
try {
$dsn = "mysql:host=localhost; dbname=discos; charset=UTF8";
$this->conexion = new PDO($dsn, 'root', '');
$this->conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
throw new Exception((string)$e->getMessage(), (int)$e->getCode());
}
}
}

```

- Crear la clase del modelo: `class ArtistaModel extends Database { ... }`

- Creamos el método para el alta: `public function alta($datos) { ... }` con las siguientes operativas:

- Extraemos los datos del array que recibimos como parámetro (solo si hemos utilizado la opción `compact` en el controlador para enviarlos al modelo)

```
extract($datos)
```

- Validar los datos recibidos del controlador (según las reglas de validación de las especificaciones).

Ejemplo:

```

if (empty($nombre) || empty($nacionalidad)) {
throw new Exception("Todos los datos son obligatorios", 10);
}

```

NOTA 1: Esta excepción la capturaremos con un `catch` de la clase `Exception` y lo que haremos es relanzarla para que llegue al controlador:

```

catch (Exception $e) {
throw new Exception($e->getMessage(), $e->getCode());
}

```

NOTA 2: Puesto que las validaciones las tendremos que realizar también en la modificación, podríamos crear un método privado para centralizarlas.

```
private function validarDatos
```

Y llamar a esta función con

```
$this->validarDatos($nombre, $nacionalidad) //this hace referencia al objeto que hemos instanciado de la clase del modelo
```

- **Confeccionamos la sentencia SQL para el INSERT**

```
INSERT INTO artista VALUES (NULL, :nombre, :nacionalidad)
```

NOTA: En este ejemplo vamos a utilizarla librería PDO de forma que utilizaremos el método `prepare` de esta librería para enlazar los valores a los parámetros de la sentencia `select` y que serán escapados automáticamente

- Realizamos el `prepare` de la sentencia

```
$stmt = $this->conexion->prepare($sql);
```

- Realizamos el bind de los parámetros. Ejemplo para el nombre:

```
$stmt->bindParam(':nombre', $nombre)
```

- Ejecutamos la sentencia con `$stmt->execute()`
- Si queremos recuperar la clave primaria asignada a la fila insertada (por ejemplo para enviarla en la respuesta al controlador)

```
$id = $this->conexion->lastInsertId()
```

- Si no hay errores confeccionamos la respuesta para que la recoja el controlador: `return array('codigo'=>'00', 'datos'=>"Artista dado de alta con el id $id")`
- Y si hay un error en la ejecución de la sentencia sql la capturaremos con un `catch` de la clase `PDOException` y lo que haremos es relanzarla para que llegue al controlador.

```
catch (PDOException $e) {
    if ($e->errorInfo[1]==1062) {
        throw new Exception('Este artista ya existe en la base de datos', 30);
    }
    } else {
        throw new Exception((string)$e->getMessage(), (int)$e->getCode());
    }
}
```

NOTA: Recordad validar el error de clave única duplicada (aunque en esta tabla artistas no se va a producir ya que la única clave única es la primaria)

EJERCICIO 2: CONSULTA DE TODOS LOS ARTISTAS

Nombre	Nacionalidad
Connie Corleone	ITA
Guns N' Roses	USA
Luca Brasi	ITA
Peter Clemenza	ITA
Rammstein	GER
Virgil Solozzo	ITA

1. Acciones a realizar en la Vista

Vamos a editar la vista **artistas.html** para añadir los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de consulta de todos los artistas.

Añadiremos el fichero **consultaartistas.js** justo antes de cerrar la etiqueta **<body>** y editaremos el fichero que ya habíamos añadido antes **inicioartista.js**

Fichero **inicioartista.js**

Añadiremos en el fichero las acciones que hay que realizar cuando se carga la vista **artista.html** para realizar la consulta de todos los artistas:

- Llamar a la función de consulta de todos los artistas **consultaArtistas()**

Fichero **consultaartistas.js**

En este fichero definiremos la función **consultaArtistas** que se encargará de realizar la petición ajax al controlador del servidor para realizar la consulta:

- Utilizaremos el controlador **artistacontroller.php**:
- En el objeto **FormData** informaremos el atributo **peticion** con 'C' de consulta: • Confeccionamos los parámetros de la petición **post**
- Realizar la llamada ajax. La respuesta será un json con todos los artistas de la tabla:

```
▼ {codigo: '00', datos: Array(6)} ⓘ
  codigo: "00"
  ▼ datos: Array(6)
    ▶ 0: {0: '10', 1: 'Connie Corleone', 2: 'ITA', id: 10}
    ▶ 1: {0: '1', 1: 'Guns N' Roses', 2: 'USA', id: 1}
    ▶ 2: {0: '9', 1: 'Luca Brasi', 2: 'ITA', id: 9}
    ▶ 3: {0: '7', 1: 'Peter Clemenza', 2: 'ITA', id: 7}
    ▶ 4: {0: '2', 1: 'Rammstein', 2: 'GER', id: 2}
    ▶ 5: {0: '8', 1: 'Virgil Solozzo', 2: 'ITA', id: 8}
```

- O bien, si el código de respuesta no es '00' un array con el mensaje de error a mostrar:

```
codigo: 42000
error: "SQLSTATE[42000]: Syntax error"
```

- Si el código de respuesta es '00' confeccionamos la tabla de artistas con la estructura siguiente:


```
<tr data-id='id del artista'><td>Nombre artista</td><th>Nacionalidad</th></tr>
```

NOTA: El atributo `data-id` de la etiqueta `tr` nos irá bien posteriormente para realizar la consulta de detalle del artista seleccionado en la tabla

- Si nos llega un array vacío del controlador `artistas.length == 0` mostramos un mensaje utilizando un `alert()`:

```
alert('No existen artistas en la base de datos')
```

- Enviamos la tabla de artistas al documento html

```
document.querySelector('#listaartistas').innerHTML = tabla
```

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador ***artistacontroller.php*** para añadir la operativa de consulta de artista.

Si la petición recibida en el controlador es una consulta

- Llamar al método del modelo que se encargará de la consulta de todos los artistas: `$respuesta =`

```
$artista->consulta()
```

- Enviamos la respuesta a la vista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo ***artistacontrol.php*** para añadir la operativa de consulta de todos los artistas.

Las modificaciones que hay que realizar son las siguientes:

- Creamos el método para la consulta: `public function consulta() { ... }` con las siguientes operativas:

- Confeccionamos la sentencia SQL

```
SELECT * FROM artista ORDER BY nombre
```

- Puesto que no hay filtro de consulta no hará falta realizar bind de parámetros por lo que podemos ejecutarla sin necesidad de realizar el `prepare`

```
$stmt = $this->conexion->query($sql)
```

- Necesitamos recuperar las filas consultadas del objeto que nos entrega el sistema gestor. Lo podemos realizar de una sola vez utilizando:

```
$artistas = $stmt->fetchAll()
```

- Si no hay errores confeccionamos la respuesta para que la recoja el controlador: `return array('codigo'=>'00', 'datos'=>$artistas)`

EJERCICIO 3: CONSULTA DEL ARTISTA SELECCIONADO EN LA TABLA



Nombre:

Nacionalidad:

Alta Artista Modificar Baja artista

Nombre	Nacionalidad
Connie Corleone	ITA

1. Acciones a realizar en la Vista

Vamos a editar la vista **artistas.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de consulta de un artista.

Para optimizar código podemos modificar el fichero **consultaartistas.js** que ya tenemos confeccionado del apartado anterior y editaremos el fichero que ya habíamos añadido antes **inicioartista.js**

Fichero **inicioartista.js**

Si nos fijamos en la plataforma a desarrollar en su conjunto observaremos que necesitaremos consultar la entidad **artistas** en tres operativas:

1. Para confeccionar la tabla de artistas en la vista **artistas.html**
2. Para realizar la consulta de detalle de un artista seleccionado en la tabla anterior
3. Para confeccionar la combo múltiple de artistas en la vista **albums.html**

Por lo tanto podríamos aprovechar la función de consulta que ya tenemos confeccionada para que tenga una salida u otra en función de un parámetro de entrada.

De esta forma vamos a modificar este fichero de inicio para:

- Llamar a la función de consulta de todos los artistas con un parámetro de entrada `consultaArtistas('T')`

NOTA: A esta función la estábamos llamando sin parámetro pero añadiremos uno para indicarle que debe confeccionar la tabla de artistas con la respuesta de la petición ajax

- Activar un listener estático sobre la tabla de artistas con la finalidad de detectar cuando pulsamos sobre una celda y poder recuperar el id asociado al artista seleccionado:

```
document.querySelector('table#listaartistas').onclick = function(event) {
  if (event.target.nodeName.toUpperCase()=='TD') {
    let id = event.target.closest('tr').getAttribute('data-id')
    consultaArtistas('F', id);
  }
}
```

NOTA 1: Estamos recuperando con `getAttribute` el valor del atributo `data-id` (que contiene la clave primaria del artista a consultar) y lo pasamos como segundo parámetro a la función que ya teníamos confeccionada `consultaArtistas()`

NOTA 2: Vemos como, en este caso, como primer parámetro pasaremos 'F' para indicar que queremos que la función cumplimente el formulario de la vista cuando reciba la respuesta de la petición

Fichero `consultaartistas.js`

En este fichero modificaremos la función `consultaArtistas` para que, aparte de confeccionar la tabla de artistas de la vista, realice la petición para la consulta de detalle de un artista y mostrar sus datos en el formulario:

- Modificaremos la definición de la función para que reciba dos parámetros (el segundo de ellos, que correspondería con el artista a consultar, opcional)

```
function consultaArtistas(salida, id=0)
```

- Utilizaremos el mismo controlador `artistacontroller.php`:

- En el objeto `FormData` informaremos el atributo `peticion` con 'C' de consulta pero añadiremos también el id del artista a consultar:

```
datos.append('idartista', id)
```

- Confeccionamos los parámetros de la petición `post`
- Realizar la llamada ajax. La respuesta será un json con el detalle del artista:

```
▼ Object 1
  codigo: "00"
  ▼ datos:
    idartista: "10"
    nacionalidad: "ITA"
    nombre: "Connie Corleone"
```

- O bien, si el código de respuesta no es '00' un array con el mensaje de error a mostrar:

```
codigo: 42000
error: "SQLSTATE[42000]: Syntax error"
```

- Si el código de respuesta es '00' trasladamos los datos del array de respuesta al formulario. Ejemplo para el nombre

```
document.querySelector('#nombre').value = mensaje.datos.nombre
```

NOTA: Recordar trasladar también el id del artista al control oculto `<input type="hidden" name="idartista" id='idartista'/>`

- Activar los botones de baja y modificación. Ejemplo para baja

```
document.querySelector('#baja').removeAttribute('disabled')
```

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador **artistacontroller.php** para añadir la operativa de consulta de un artista.

Si la petición recibida en el controlador es una consulta (que será el mismo tipo de petición que la que hemos realizado en la consulta de todos los artistas)

- Recuperamos el id del libro (que vendrá sin informar en caso que la consulta sea de todos los artistas)

```
$idartista = $_POST['idartista']
```

- Lo compactamos en un array con `$datos = compact('idartista')` (o bien enviamos el id como dato)

- Modificar la llamada al método del modelo para enviar el id

```
$respuesta = $artista->consulta($datos)
```

- Enviamos la respuesta a la vista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo **artistamodel.php** para añadir la operativa de consulta de un artista

Utilizaremos el mismo método de consulta que hemos creado para la consulta de todos los artistas al cual añadiremos las siguientes modificaciones:

- Ahora el método recibirá un parámetro de entrada **function consulta(\$datos)** que tendremos que extraer con **extract(\$datos)** (si hemos utilizado **compact** en el controlador)

- Si el id del artista está informado (es mayor que 0) confeccionamos la sentencia SQL **SELECT * FROM artista WHERE idartista=\$idartista**

- Ejecutamos la sentencia (no hace falta realizar bind para el filtro al ser la clave primaria) **\$stmt = \$this->conexion->query(\$sql)**

- Recuperamos las filas consultadas del objeto que nos entrega el sistema gestor. En esta consulta obtendremos, como máximo, una fila por lo que nos iría bien obtener un array asociativo de una dimensión. Para ello:

```
$stmt->setFetchMode(PDO::FETCH_ASSOC)  
$artistas = $stmt->fetch()
```

NOTA: Si no definimos el tipo de extracción de datos con **setFetchMode** obtendremos el siguiente array con claves escalares y asociativas para los mismos datos.

```
0: "10"  
1: "Connie Corleone"  
2: "ITA"  
idartista: "10"  
nacionalidad: "ITA"  
nombre: "Connie Corleone"
```

- Si no hay errores confeccionamos la respuesta para que la recoja el controlador: **return array('codigo'=>'00', 'datos'=>\$artistas)**

EJERCICIO 4: MODIFICACION DE ARTISTA



1. Acciones a realizar en la Vista

Vamos a editar la vista **artistas.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de modificación de un artista.

Fichero **inicioartista.js**

Modificaremos este fichero para añadir las acciones que hay que realizar cuando se carga la vista **artista.html** y que, para el caso de la modificación, son las siguientes:

- Activar el listener para detectar la pulsación del botón de modificación de artista

document.querySelector('#modificar').onclick = modificarArtista

Fichero **modificarartista.js**

Crearemos este fichero donde definiremos la función **modificarArtista** que se encargará de realizar la petición ajax al controlador del servidor para realizar la modificación:

- Recuperar los datos del formulario.
- Seguiremos utilizando el controlador **artistacontroller**:
- Confeccionar el objeto **FormData** con los datos de la petición y petición 'M' de modificación: •

Confeccionar el objeto con los parámetros de la petición

- Realizar la llamada ajax. La respuesta será un json con la estructura que ya hemos visto: **['codigo' =>**

99, 'datos' => 'datos o mensaje de respuesta', 'error' => 'mensaje de error si lo hay']

- Si el código de respuesta es '00' mostramos el mensaje de respuesta en un alert y volveremos a llamar a la función de consulta de artistas para actualizar la tabla con los datos que acabamos de modificar

```
consultaArtistas('T')
```

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador **artistacontroller.php** para añadir la operativa de modificación de artista.

Las acciones que hay que añadir para el tipo de petición 'M' son las siguientes: • Recuperamos los datos que nos llegan de la vista:

- Compactamos los datos en un array

```
$datos = compact('idartista', 'nombre', 'nacionalidad')
```

- Llamar al método del modelo que se encargará de la modificación del artista:

```
$artista->modificacion($datos)
```

- Confeccionamos la respuesta a la vista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo **artistacontrol.php** para añadir la operativa de modificación de artista.

Las acciones que hay que realizar son las siguientes:

- Creamos el método para la modificación: `public function modificacion($datos) { ... }` con las siguientes operativas:

- Extraemos los datos del array que recibimos como parámetro
- Validar los datos recibidos del controlador
- Confeccionamos la sentencia SQL para el **UPDATE**

```
UPDATE artista SET nombre = :nombre, nacionalidad = :nacionalidad WHERE idartista = $idartista
```

- Realizamos el prepare de la sentencia

```
$stmt = $this->conexion->prepare($sql);
```

- Realizamos el bind de los parámetros.
- Ejecutamos la sentencia con `$stmt->execute()`

- Podemos validar si se ha modificado alguna fila utilizando `$stmt->rowCount() == 0`. En caso que no se haya modificado ninguna puede ser debido a que el id del artista no existe en la tabla o bien que el usuario ha pulsado el botón de modificar sin realizar ningún cambio en los datos del formulario
- Si no hay errores confeccionamos la respuesta para que la recoja el controlador: `return array('codigo'=>'00', 'datos'=>"Artista dado de alta con el id $id")`

EJERCICIO 5: BAJA DE ARTISTA



1. Acciones a realizar en la Vista

Vamos a editar la vista **artistas.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de baja de un artista.

Fichero **inicioartista.js**

Modificaremos este fichero para añadir las acciones que hay que realizar cuando se carga la vista **artista.html** y que, para el caso de la baja, son las siguientes:

- Activar el listener para detectar la pulsación del botón de modificación de artista

`document.querySelector('#baja').onclick = bajaArtista`

Fichero **bajaartista.js**

Crearemos este fichero donde definiremos la función **bajaArtista** que se encargará de realizar la petición ajax al controlador del servidor para realizar la baja:

- Recuperar el id del artista a borrar.
- Seguiremos utilizando el controlador **artistacontroller**:
- Confeccionar el objeto **FormData** con el id del artista y petición 'B' de baja:
- Confeccionar el objeto

con los parámetros de la petición

- Realizar la llamada ajax. La respuesta será un json con la estructura que ya hemos visto: ['codigo' =>

99, 'datos' => 'datos o mensaje de respuesta', 'error' => 'mensaje de error si lo hay'] • Si el código de

respuesta es '00':

- Mostramos el mensaje de respuesta en un alert
- Reiniciamos el formulario
- Desactivamos los botones de baja y modificación
- Volveremos a llamar a la función de consulta de artistas
- Si el código de respuesta no es '00' mostramos el mensaje de error

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador **artistacontroller.php** para añadir la operativa de baja de artista.

Las acciones que hay que añadir para el tipo de petición 'B' son las siguientes: • Recuperamos el id del artista a borrar:

- Compactamos el dato en un array
- Llamar al método del modelo que se encargará de la baja del artista:

`$artista->baja($datos)`

- Confeccionamos la respuesta a la vista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo **artistamodel.php** para añadir la operativa de baja de artista. Las acciones que hay que realizar son las siguientes:

- Creamos el método para la baja: `public function baja($datos) { ... }` con las siguientes operativas:

- Extraemos el dato del array que recibimos como parámetro
- Validar el id recibido del controlador
- Confeccionamos la sentencia SQL para el **DELETE**

`DELETE FROM artista WHERE idartista = $idartista`

- Ejecutamos directamente la sentencia con `$stmt = $this->conexion->query($sql)`

- Podemos validar si se ha borrado alguna fila utilizando `$stmt->rowCount() == 0`. En caso que no se haya borrado ninguna lanzaremos una excepción con el mensaje 'Artista no existe'
- Si no hay errores confeccionamos la respuesta para que la recoja el controlador: `return array('codigo'=>'00', 'datos'=>"Artista borrado en la base de datos")`

ALTERNATIVA DE DISEÑO DEL CONTROLADOR

Tal como hemos visto hemos diseñado el controlador con la siguiente estructura:

- *Recuperar y validar tipo de petición ('A', 'C', 'M' o 'B')*
- *Instanciar un objeto del modelo*
- *Evaluar tipo de petición para:*
 - *Recuperar los datos que correspondan a cada tipo*
 - *Compactarlos en un array*
- *Llamar al método correspondiente del modelo y recoger la respuesta*
- *Enviar la respuesta a la vista*

Pero, en el caso que no tengamos que realizar operativas adicionales con la respuesta del modelo, podemos simplificar la estructura y centralizar las llamadas a los métodos del modelo en un solo punto del programa utilizando la función `call_user_func()`. Nuestro controlador quedaría así:

- *Incorporar modelo*
- *Recuperar y validar tipo de petición ('A', 'C', 'M' o 'B')*
- *Instanciar un objeto del modelo al que llamaremos `$objeto`*
- *Evaluar tipo de petición para:*
 - *Recuperar los datos que correspondan a cada tipo*
 - *Compactarlos en un array que llamaremos `$datos`*
 - *Informar en una variable `$metodo` el método del modelo a llamar*
- *Llamar al método correspondiente del modelo y recoger la respuesta de forma dinámica utilizando:*
`$respuesta = call_user_func(array($objeto, $metodo), $datos);`
- *Enviar la respuesta a la vista*

EJERCICIO 6: ALTA DE ALBUM

Discográfica El disco cuadrado

Título:

Año:

Género:

Artista/s:

Buscar:

Título	Año
C'era una volta in occidente	2000
Ciao Bambina	2000
Dåren i båten	2000
Du Hast	1997
Flyger över händelsehorisonten	2000

1. Acciones a realizar en la Vista

Vamos a editar la vista **albums.html** para añadir los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de alta.

Añadiremos dos ficheros justo antes de cerrar la etiqueta **<body>**

1. altaalbum.js
2. consultagenero.js
3. consultaartista.js
4. inicioalbum.js

Fichero **inicioalbum.js**

En este fichero indicaremos las acciones que hay que realizar cuando se carga la vista **albums.html** y que, para el caso del alta, son las siguientes:

- Activar el listener para detectar la pulsación del botón de alta
- Llamar a la función que se encargará de informar de forma dinámica la combo de géneros musicales

```
consultaGeneros()
```

- Llamar a la función que se encargará de informar de forma dinámica la combo múltiple de artistas

```
consultaArtistas('C') //el parámetro 'C' es para indicar a la función que confeccione una combo
```

Fichero *altaalbum.js*

En este fichero definiremos la función **altaAlbum** que se encargará de realizar la petición ajax al controlador del servidor para realizar el alta:

- Recuperar los datos del formulario (título, año y género se recuperan utilizando el mismo procedimiento que hemos visto en alta de artista).

- Para recuperar los artistas a asociar al álbum, puesto que tenemos una select multiple, hay que hacerlo de la siguiente forma:

- Guardamos en un objeto todas las opciones de la combo seleccionadas

```
let artistas = document.querySelectorAll('#artistas option:checked')
```

- Creamos un array vacío para guardar los id de los artistas seleccionados

```
arrayArtistas = []
```

- Recorremos el objeto con las opciones seleccionadas para extraer el id del artista de cada una de ellas (vamos a utilizar el potente bucle **forEach**)

```
artistas.forEach((item) => {  
  arrayArtistas.push(item.value)  
})
```

- Informar al controlador donde vamos a realizar la petición y que será **albumcontroller**.

- Confeccionar el objeto **FormData** de la petición. Vamos a seguir el mismo esquema que hemos utilizado para los artistas. Es decir, enviaremos los datos junto con un atributo 'peticion' con el valor 'A':

El array con los id de los artistas a asociar lo enviaremos en la petición en formato json de la siguiente manera:

```
datos.append('artistas', JSON.stringify(arrayArtistas))
```

- Confeccionar el objeto con los parámetros de la petición
- Realizar la llamada ajax al controlador. La respuesta será un json con el formato que ya hemos visto en artistas

['codigo' => 99, 'datos' => 'datos o mensaje de respuesta', 'error' => 'mensaje de error si lo hay'] • Si el código

de respuesta es '00':

- Reiniciamos el formulario
- Desactivamos los botones de baja y modificación.
- Limpiamos el filtro de búsqueda (que veremos más adelante)

`document.querySelector('#buscar').value = ''`

- Reiniciamos la variable global de paginación: `pag = 1`

◦ Y, puesto que el alta que acabamos de efectuar debe reflejarse en la tabla de consulta, tendremos que realizar una llamada a la función javascript que se encargará de confeccionarla

`consultaAlbums()`

Fichero `consultaartista.js`

Modificaremos este fichero (que ya habíamos creado en la operativa de artistas) para que pueda atender el tipo de petición 'C' de forma que nos confeccione una combo una vez recibida la respuesta del servidor:

`<option value='id del artista'>nombre del artista</option>`

Y que enviaremos a:

`document.querySelector('#artistas').innerHTML = opciones`



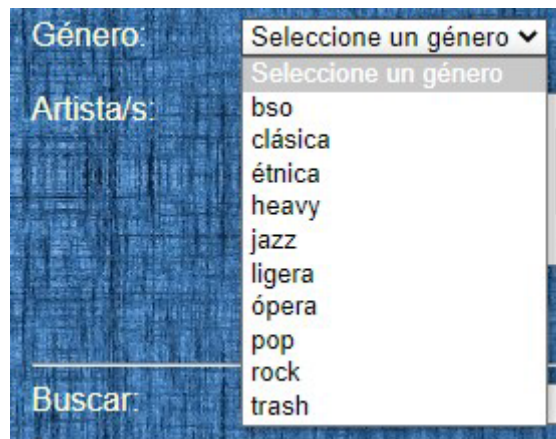
Fichero `consultagenero.js`

Tendremos que crear un nuevo fichero para realizar una petición al servidor para traernos todos los géneros de la tabla **genero** (en el fichero de recursos veréis que los géneros se encuentran informados de forma estática pero, idealmente, tendríamos que recuperarlos de la base de datos)

`<option value='id del genero'>nombre del genero</option>`

Y que enviaremos a:

`document.querySelector('#genero').innerHTML = opciones`



2. Acciones a realizar en el Controlador Album

Vamos a confeccionar el controlador ***albumcontroller.php*** para añadir la operativa de alta de albums.

NOTA: Vamos a utilizar la misma estructura que ya hemos visto para el controlador de la operativa de artistas. Las acciones que hay que realizar son las siguientes:

- Incorporar el fichero que utilizaremos como modelo

`require '../models/albummodel.php';` //recordad que se encuentra en la carpeta models • Recuperar

el tipo de petición que nos llega de la vista

- Instanciamos un objeto del modelo:

`$artista = new AlbumModel();`

- Evaluamos la petición recibida y, en caso que sea un alta:

- Recuperamos los datos para el alta y compactarlos

NOTA: Recordad que para los artistas a asociar vamos a recibir un json con la relación de id y que tendremos que convertir a array php utilizando

`$artistas = json_decode($_POST['artistas']);`

- Llamar al método del modelo que se encargará del alta :

`$respuesta = $album->alta($datos)`

- Confeccionamos la respuesta a la vista (que va a ser siempre un json):

`echo json_encode($respuesta);`

3. Acciones a realizar en el Modelo Album

Vamos a confeccionar el modelo *albummodel.php* para añadir la operativa de alta. Las acciones que hay que realizar son las siguientes:

- Incorporar el fichero de conexión a la base de datos:
- Crear la clase del modelo: `class AlbumModel extends Database { ... }`
- Creamos el método para el alta: `public function alta($datos) { ... }` con las siguientes operativas:
 - Extraemos los datos del array que recibimos como parámetro (solo si hemos utilizado la opción *compact* en el controlador para enviarlos al modelo)
 - Validar los datos recibidos del controlador (según las reglas de validación de las especificaciones).

NOTA: Para validar que nos llega informado, como mínimo un artista, podemos utilizar:

```
if (sizeof($artistas) == 0) { //sizeof() nos devuelve la longitud de un array
throw new Exception("Se debe seleccionar al menos un artista", 10);
}
```

- Ahora tenemos que realizar dos INSERT:

1. Una fila en la tabla *album* con los datos del disco a dar de alta
 2. Tantas filas como artistas hemos seleccionado en la combo, en la tabla de relación *relartistaalbum* (recordad que hay una relación muchos a muchos)
- Para asegurar la integridad referencial en la base de datos tenemos que garantizar que se actualizan las dos tablas o ninguna y, para ello, activaremos una transacción con:

```
$this->conexion->beginTransaction()
```

NOTA: Imaginad que el INSERT en la tabla *album* se realiza correctamente y que, por el motivo que sea, falla el INSERT en la tabla de relación. Tendríamos una fila correspondiente a un álbum del que no sabríamos a qué artista o artistas pertenece

- Confeccionamos la sentencia SQL para el *INSERT* de la tabla *album*
- Realizamos el prepare de la sentencia, el bind de los parámetros y su ejecución
- Necesitamos recuperar la clave primaria asignada a la fila insertada para poder realizar los INSERT en la tabla de relación

```
$id = $this->conexion->lastInsertId()
```

- Para dar de alta las relaciones tenemos dos opciones:
 1. Si la operativa es de alta insertamos en *relartistaalbum* tantas filas como id de artistas tengamos en el array de datos que recibimos del controlador
 2. O bien, creamos una función que sirva tanto para el alta como para la modificación y que se encargue de borrar todas las relaciones cuya clave sea la del álbum creado o modificado y las vuelva

a crear desde cero

Podéis utilizar el sistema que consideréis oportuno pero en este ejemplo vamos a considerar el segundo:

- Borrarnos todas las relaciones de **relartistaalbum** cuya clave sea la id del album: **DELETE FROM relartistaalbum WHERE idalbum = \$idalbum**

- Recorremos el array con los id de los artistas a asignar y, para cada uno de ellos, creamos una fila en la tabla **relartistaalbum**

INSERT INTO relartistaalbum VALUES (\$idartista, \$idalbum)

- Si no hay errores finalizamos exitosamente la relación con **\$this->conexion->commit()** y confeccionamos la respuesta para que la recoja el controlador:

return array('codigo'=>'00', 'datos'=>"Album dado de alta con el id \$id")

- Y si hay un error en la ejecución de la operativa la capturaremos, la relanzamos para que llegue al controlador y finalizamos la transacción con **\$this->conexion->rollback()**

4. Acciones a realizar en el Controlador Genero

Vamos a confeccionar el controlador **generocontroller.php** para añadir la operativa de consulta de generos.

NOTA: Vamos a utilizar la misma estructura que ya hemos visto para el controlador de la operativa de artistas y de albums pero, para este ejercicio, solo necesitaremos la opción de Consulta de todos los géneros de la tabla

5. Acciones a realizar en el Modelo Genero

Vamos a confeccionar el modelo **generomodel.php** para añadir la operativa de consulta de géneros.

NOTA: Vamos a utilizar la misma estructura que ya hemos visto para el modelo de la operativa de artistas y de albums pero, para este ejercicio, solo necesitaremos la opción de Consulta de todos los géneros de la tabla

```
▼ 1: Array(10)
  ►0: {0: '8', 1: 'bso', idgenero: '8', genero: 'bso'}
  ►1: {0: '5', 1: 'clásica', idgenero: '5', genero: 'clásica'}
  ►2: {0: '9', 1: 'étnica', idgenero: '9', genero: 'étnica'}
  ►3: {0: '6', 1: 'heavy', idgenero: '6', genero: 'heavy'}
  ►4: {0: '4', 1: 'jazz', idgenero: '4', genero: 'jazz'}
  ►5: {0: '7', 1: 'ligera', idgenero: '7', genero: 'ligera'}
  ►6: {0: '11', 1: 'ópera', idgenero: '11', genero: 'ópera'}
  ►7: {0: '1', 1: 'pop', idgenero: '1', genero: 'pop'}
  ►8: {0: '2', 1: 'rock', idgenero: '2', genero: 'rock'}
  ►9: {0: '10', 1: 'trash', idgenero: '10', genero: 'trash'}
```


EJERCICIO 7: CONSULTA DE TODOS LOS ALBUMS

Título	Año
C'era una volta in occidente	2000
Ciao Bambina	2000
Dåren i båten	2000
Du Hast	1997
Flyger över händelsehorisonten	2000

1. Acciones a realizar en la Vista

Vamos a editar la vista **albums.html** para añadir los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de consulta de todos los albums.

Añadiremos el fichero **consultaalbums.js** justo antes de cerrar la etiqueta **<body>** y editaremos el fichero que ya habíamos añadido antes **inicioartista.js**

Fichero **inicioalbum.js**

Añadiremos en el fichero las acciones que hay que realizar cuando se carga la vista **album.html** para realizar la consulta de todos los albums:

- Llamar a la función de consulta de todos los artistas **consultaAlbums()**

Fichero **consultaalbums.js**

En este fichero definiremos la función **consultaAlbums** que se encargará de realizar la petición ajax al controlador del servidor para realizar la consulta:

- Utilizaremos el controlador **albumcontroller.php**:

- Seguiremos exactamente el mismo procedimiento que el utilizado para artistas pero, en este caso, no será necesario el parámetro que nos indicaba el tipo de salida que queríamos con los datos obtenidos del servidor ya que ésta (o informar la tabla de albums o informar el formulario de mantenimiento) la determinaremos según el valor del parámetro opcional **id** que recibirá la función.

Ejemplo:

```
function consultaAlbums(id=0)
```

Si id=0 estamos consultando todos los albums y la salida será, por tanto, la tabla Si id>0 estamos consultando el detalle de un álbum y la salida será el formulario

- Si el código de respuesta es '00' confeccionamos la tabla de albums con la estructura siguiente:

```
<tr data-id='id del album'><td>Título album</th><th>Año</th></tr>
```

2. Acciones a realizar en el Controlador

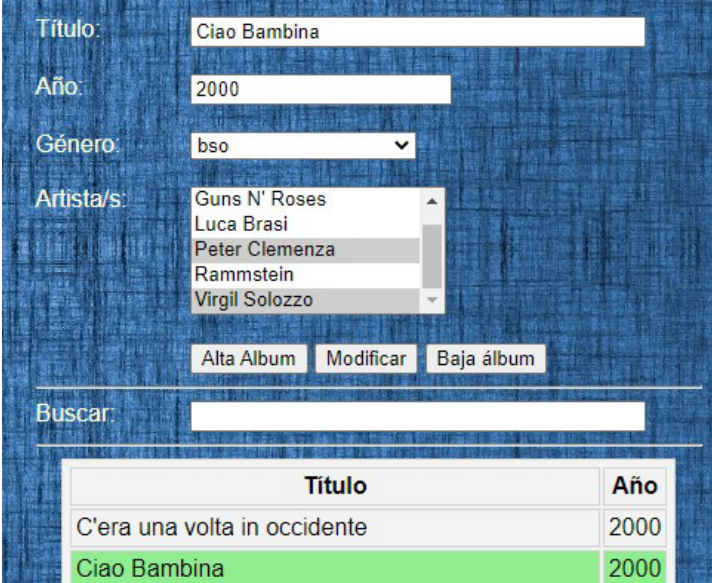
Vamos a modificar el controlador **albumcontroller.php** para añadir la operativa de consulta Utilizaremos exactamente el mismo procedimiento visto para la tabla artista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo **albummodel.php** para añadir la operativa de consulta de todos los albums.

Utilizaremos exactamente el mismo procedimiento visto para la tabla artista

EJERCICIO 8: CONSULTA DEL ALBUM SELECCIONADO EN LA TABLA



Título:

Año:

Género:

Artista/s:

Buscar:

Título	Año
C'era una volta in occidente	2000
Ciao Bambina	2000

1. Acciones a realizar en la Vista

Vamos a editar la vista **albums.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de consulta de un album.

Fichero **inicioalbum.js**

Vamos a modificar este fichero de inicio para:

- Activar un listener estático sobre la tabla de albums con la finalidad de detectar cuando pulsamos sobre una celda y poder recuperar el id asociado al album seleccionado:

```
document.querySelector('table#listaalbums').onclick = function(event) {
  if (event.target.nodeName.toUpperCase()=='TD') {
    let id = event.target.closest('tr').getAttribute('data-id')
    consultaAlbums(id);
  }
}
```

NOTA 2: Vemos como, en este caso, estamos pasando como parámetro el id del album seleccionado en la tabla Fichero

consultaalbums.js

En este fichero tendremos que realizar las mismas acciones que las vistas para la tabla artistas pero con alguna novedad que os indico más adelante en este apartado:

- Recoger el parámetro id que nos llegará de la petición al seleccionar una fila en la tabla • Realizar la petición al controlador **artistacontroller** con la opción 'C'
- La respuesta será un array con tantas filas como artistas tenga asignados el álbum. Por ejemplo:

```
▼ (2) ['00', Array(2)] ⓘ
  0: "00"
  ▼ 1: Array(2)
    ▼ 0:
      idalbum: "30"
      idartista: "2"
      idgenero: "6"
      titulo: "Flyger över händelsehorisonten"
      year: "2000"
      ► [[Prototype]]: Object
    ▼ 1:
      idalbum: "30"
      idartista: "9"
      idgenero: "6"
      titulo: "Flyger över händelsehorisonten"
      year: "2000"
```

- Con la respuesta informar los controles del formulario.

Los datos correspondientes a id del álbum, título, id del género y título los informaremos tal como estamos ya acostumbrados. Ejemplo para id del género:

```
document.querySelector('#genero').value = album.idgenero
```

Pero ¿cómo seleccionamos las opciones correspondientes a los artistas en las que puede haber uno o más de uno? Tenemos dos opciones (una de ellas tiene claramente una ventaja determinante sobre la otra. Intentad averiguar cual)

Opción estática

- Guardar en un array de objetos todas las opciones de la combo

```
let opcionesArtistas = document.querySelectorAll('#artistas option')
```

- Recorrer el array de opciones (por ejemplo con **forEach**) y, para cada una de ellas guardar el contenido del atributo **value** (que será el id del artista) en un array escalar

```
opcionesArtistas.forEach((opcion) => {  
  arrayArtistas.push(opcion.value)  
})
```

- Hay que desactivar la opción o opciones que estuvieran seleccionadas de una consulta anterior. Eliminamos el atributo **selected** de la opción/es cuyo atributo **value** coincida con el id del artista del álbum

```
document.querySelector(`#artistas option[value='${opcion.value}']`).removeAttribute('selected')
```

- Y, por último, hay que verificar si el/los id de artista que tendremos en el mensaje de respuesta del servidor se encuentran dentro del array de opciones

```
arrayArtistas.includes(album[i].idartista)
```

- Y, de ser así, marcamos la opción de la combo como seleccionada

```
document.querySelector(`#artistas option[value='${album[i].idartista}']`).setAttribute('selected', true)
```

Opción dinámica

- Volver a renderizar la combo de artistas con una llamada a la función

consultaArtistas('C'). Esto nos permite volver a inicializar la combo eliminando las posibles opciones seleccionadas de una consulta anterior (recordad que estamos trabajando con llamadas asíncronas que no refrescan la página)

- Recorrer el array de respuesta del servidor con los datos del álbum (habrá una fila para cada artista) y, para cada fila, activar el atributo **selected** de la opción cuyo atributo **value** coincida con el id del artista del álbum

```
for (i in album) {
```

```
let idartista = album[i].idartista
document.querySelector(`#artistas option[value='${idartista}']`).setAttribute('selected', true) }
```

◦ ¿Funciona?... Pues hemos seguido los pasos correctamente veremos como se renderiza la combo de nuevo, se realiza la consulta del álbum seleccionado pero no se marcan las opciones que correspondan a los artistas de álbum

◦ ¿Por qué?... Pues porque la llamada a **consultaArtistas** es una petición asíncrona al servidor de forma que el código que venga a continuación de esta llamada (recorrer el array de albums para marcar la opción del artista) se ejecuta antes de finalice la petición ajax.

◦ ¿Podemos hacer algo?... Pues sí, vamos a indicar al navegador que no ejecute el código de marcar la opción del artista hasta que no se complete la petición ajax:

Marcamos la función **consultaArtistas** como asíncrona

```
async function consultaArtistas(salida, id=0)
```

Dentro de esta función indicamos que instrucción tenemos que esperar que se complete (obviamente es el **fetch** que es la que realiza la petición al servidor)

```
await fetch(servicio, parametros)
```

◦ En nuestro fichero **consultaalbum.js** indicamos que el código para marcar la opción u opciones del artista del álbum se ejecute una vez completada la función

```
consultaArtistas
```

```
consultaArtistas('C')
.then(function() {
  for (i in album) {
    let idartista = album[i].idartista
    document.querySelector(`#artistas option[value='${idartista}']`).setAttribute('selected', true)
  }
})
```

- Activar los botones de baja y modificación.

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador **albumcontroller.php** para añadir la operativa de consulta de detalle de un álbum

Utilizaremos exactamente el mismo procedimiento visto para la tabla artista

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo **albummodel.php** para añadir la operativa de consulta de detalle de un álbum.

Utilizaremos un procedimiento parecido al que hemos visto para la tabla artista pero con algunas modificaciones

- Necesitamos los datos **título**, **año** y **id** del género (que tenemos en la tabla **albums**) y la relación de id de los artistas a los que pertenece el álbum (que tenemos en la tabla de relación)

Para recuperar la info de las dos tablas en una sola consulta nos iría muy bien utilizar un inner Join

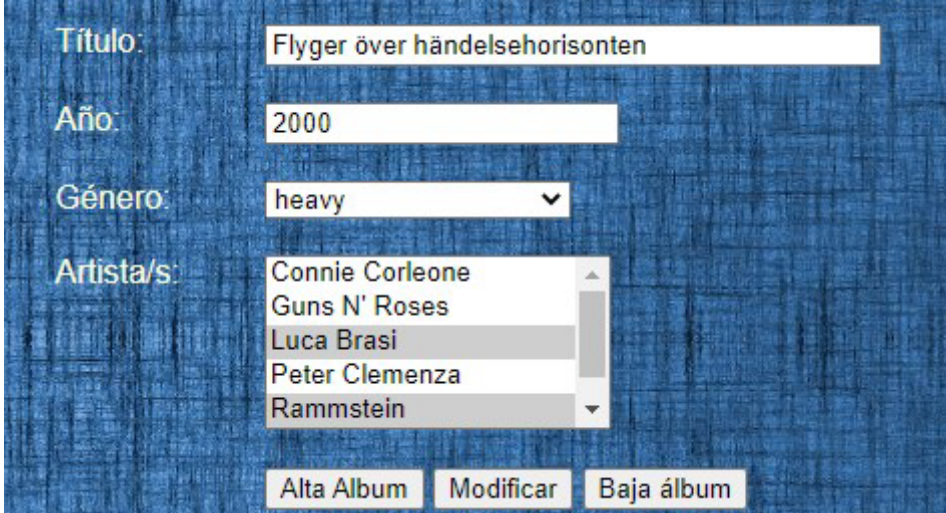
```
SELECT a.idalbum, titulo, year, idgenero, b.idartista
FROM album a INNER JOIN relartistaalbum b
ON a.idalbum = b.idalbum
WHERE a.idalbum = $idalbum
```

Ejemplo de respuesta para un album con dos artistas

```
▼ (2) ['00', Array(2)] ⓘ
  0: "00"
  ▼ 1: Array(2)
    ▼ 0:
      idalbum: "24"
      idartista: "7"
      idgenero: "8"
      titulo: "Ciao Bambina"
      year: "2000"
      ► [[Prototype]]: Object
    ▼ 1:
      idalbum: "24"
      idartista: "8"
      idgenero: "8"
      titulo: "Ciao Bambina"
      year: "2000"
```

- El resto del procedimiento es similar al de artista

EJERCICIO 9: MODIFICACION DE ALBUM



Título: Flyger över händelsehorisonten

Año: 2000

Género: heavy

Artista/s: Connie Corleone
Guns N' Roses
Luca Brasi
Peter Clemenza
Rammstein

Alta Album Modificar Baja álbum

1. Acciones a realizar en la Vista

Vamos a editar la vista **albums.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de modificación de un album.

Fichero **inicioalbum.js**

Modificaremos este fichero para añadir las acciones que hay que realizar cuando se carga la vista **album.html** y que, para el caso de la modificación, son las siguientes:

- Activar el listener para detectar la pulsación del botón de modificación de álbum

Fichero **modificaralbum.js**

Crearemos este fichero donde definiremos la función **modificarAlbum** que se encargará de realizar la petición ajax al controlador del servidor para realizar la modificación:

- Recuperar los datos del formulario.
- Seguiremos utilizando el controlador **albumcontroller**:
- Confeccionar el objeto **FormData** con los datos de la petición y petición 'M' de modificación: •

Confeccionar el objeto con los parámetros de la petición

- Realizar la llamada ajax. La respuesta será un json con la estructura que ya hemos visto:
- Si el código de respuesta es '00' mostramos el mensaje de respuesta en un alert y volveremos a llamar a la función de consulta de albums para actualizar la tabla con los datos que acabamos de modificar

`consultaAlbums()`

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador ***albumcontroller.php*** para añadir la operativa de modificación.

Utilizaremos exactamente el mismo procedimiento visto para el alta de album. Recordad que, para modificación nos llegarán:

- EL id del álbum a modificar
- Un json con los artistas a asignar al álbum y que tendremos que convertir a array con: `$artistas =`

`json_decode($_POST['artistas']);`

3. Acciones a realizar en el Modelo

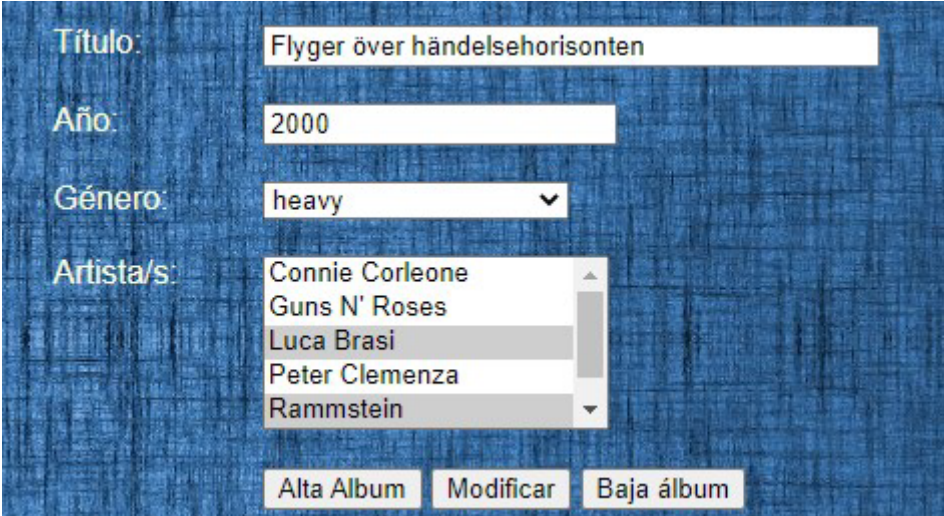
Vamos a modificar el modelo ***albummodel.php*** para añadir la operativa de modificación.

Utilizaremos exactamente el mismo procedimiento visto para el alta de álbum. • Crear el método

para la modificación: `public function modificacion($datos)` • Extraer los datos del array que recibimos

como parámetro

- Validar los datos recibidos del controlador
- Iniciar modo transaccional
- Confeccionar la sentencia SQL para el **UPDATE**, realizar el bind de parámetros y ejecutarla
- Actualizar la tabla de relación borrando todas las filas para el album modificado y volviendo a crear las relaciones desde cero
- Si no hay errores realizamos el **commit** y confeccionamos la respuesta para que la recoja el controlador:

EJERCICIO 10: BAJA DE ALBUM

Título: Flyger över händelsehorisonten

Año: 2000

Género: heavy

Artista/s: Connie Corleone
Guns N' Roses
Luca Brasi
Peter Clemenza
Rammstein

Alta Album Modificar Baja álbum

1. Acciones a realizar en la Vista

Vamos a editar la vista **albums.html** para añadir o modificar los ficheros **javascript** necesarios para realizar las peticiones asíncronas al servidor que correspondan a la operativa de baja de un album.

Fichero **inicioalbum.js**

Modificaremos este fichero para añadir las acciones que hay que realizar cuando se carga la vista **album.html** y que, para el caso de la modificación, son las siguientes:

- Activar el listener para detectar la pulsación del botón de baja de álbum Fichero **bajaalbum.js**

Crearemos este fichero donde definiremos la función **bajaAlbum** que se encargará de realizar la petición ajax al controlador del servidor para realizar la baja:

- Recuperar el id del álbum del formulario.
- Seguiremos utilizando el controlador **albumcontroller**:
- Confeccionar el objeto **FormData** con los datos de la petición y petición 'B' de baja: • Confeccionar el objeto con los parámetros de la petición
- Realizar la llamada ajax. La respuesta será un json con la estructura que ya hemos visto: • Si el código de respuesta es '00':
 - Mostramos el mensaje de respuesta en un alert

- Reiniciar el formulario
- Desactivar los botones de baja y modificación
- Volveremos a llamar a la función de consulta de albums para actualizar la tabla

2. Acciones a realizar en el Controlador

Vamos a modificar el controlador ***albumcontroller.php*** para añadir la operativa de baja. Utilizaremos exactamente el mismo procedimiento visto para baja de artista.

3. Acciones a realizar en el Modelo

Vamos a modificar el modelo ***albummodel.php*** para añadir la operativa de baja. Utilizaremos exactamente el mismo procedimiento visto para baja de artista. • Crear el método para la baja: **public**

function baja(\$datos)

- Extraer los datos del array que recibimos como parámetro
- Validar los datos recibidos del controlador
- Confeccionar la sentencia SQL para el **DELETE** y ejecutarla
- Validar que se ha borrado una fila
- SI no hay errores Confeccionamos la respuesta para que la recoja el controlador

NOTA: No hará falta borrar las relaciones de **relartistaalbum** porque el diseño de la base de datos contiene una restricción de tipo **CASCADE** entre ambas tablas de forma que, al borrar un álbum, el sistema gestor borrará automáticamente las filas de la tabla de relación