Tema 7: Acceso a base de datos MySQL

Ciclo Superior DAW

Asignatura: Desarrollo web en entorno servidor

Curso 20/21

Introducción

- En este capítulo veremos los siguientes conceptos:
  - Tecnologías que permiten el acceso a la información disponible en almacenes de datos.
  - Crear aplicaciones que establezcan conexiones con bases de datos.
  - Recuperar información de una base de datos y visualizarla en una página web.
  - Crear aplicaciones web que permitan la actualización y la eliminación de información disponible en una base de datos.
  - Usar transacciones para mantener la consistencia de la información.

Acceso a base de datos

- **Una API** (Interfaz de Programación de Aplicaciones) **define las clases, métodos, funciones y variables que la aplicación necesita emplear para realizar una tarea concreta.**
- En el caso de aplicaciones de PHP que necesiten acceder a una base de datos, las APIs necesarias son extensiones de PHP.

Acceso a base de datos

- PHP soporta más de 15 sistemas gestores de bases de datos: SQLite, Oracle, SQL Server, PostgreSQL, MySQL, etc.
- Hasta la versión 5 de PHP, el acceso a las bases de datos se hacía principalmente utilizando extensiones específicas para cada sistema gestor de base de datos (extensiones nativas).

Acceso a base de datos

- Es decir, si queríamos acceder a una base de datos de PostgreSQL, deberíamos instalar y utilizar la extensión de ese gestor en concreto. Las funciones y objetos a utilizar eran distintos para cada extensión.

Acceso a base de datos

- A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: **PDO**.
- La gran ventaja de PDO está clara: podemos seguir utilizando una misma sintaxis, aunque cambiemos el motor de nuestra base de datos.
- Por el contrario, en algunas ocasiones preferiremos seguir usando extensiones nativas en nuestros programas, pues ofrecen más potencia.

Acceso a base de datos

- De los distintos SGBD existentes **vamos a trabajar con MySQL.**
- **MySQL es un gestor de bases de datos relacionales de código abierto bajo licencia GNU GPL.**
- **Es el gestor de bases de datos más empleado con el lenguaje PHP.**

Operaciones básicas de manipulación datos SQL

- Las cuatro instrucciones básicas para trabajar con bases de datos son:
  - **SELECT**: muestra información sobre los datos almacenados en la base de datos.
  - **INSERT**: Inserta filas en una tabla.
  - **UPDATE**: Actualiza información de una tabla.
  - **DELETE**: Borra filas de una tabla.

Operaciones básicas de manipulación datos SQL. SELECT

- La sintaxis básica de una consulta SELECT es:

```
SELECT [ALL / DISTINCT] [*] / [Campos] AS [Expresion]
FROM Tabla
WHERE Condiciones
ORDER BY ListaColumnas [ASC / DESC]
```

Operaciones básicas de manipulación datos SQL. SELECT

- **ALL / DISTINCT**
    - **ALL**: es el valor predeterminado.
    - **DISTINCT**: especifica que el conjunto de resultados sólo puede incluir filas únicas.
- **Nombres de campos**
    - Tenemos una lista de nombres de campos de la tabla que queremos devolver, separados por comas. Si queremos que nos devuelva todos los campos de la tabla **utilizamos el "*".**

Operaciones básicas de manipulación datos SQL. SELECT

- **AS**
    - Permite renombrar columnas si lo utilizamos en la cláusula SELECT, o renombrar tablas si lo utilizamos en la cláusula FROM. Es opcional.
- **FROM**
    - Indica las tablas en las que vamos a hacer la consulta.

Operaciones básicas de manipulación datos SQL. SELECT

- **WHERE**
    - Filtra las filas que se van a devolver. Se devuelven sólo las que cumplen ciertas condiciones.
- **ORDER BY**
    - Establece el orden de las filas del conjunto de resultados.

Operaciones básicas de manipulación datos SQL. SELECT

- **ASC / DESC**
    - Define si los resultados se ordenarán de forma ascendente o descendente.

Operaciones básicas de manipulación datos SQL. SELECT

Algunos ejemplos:

```
SELECT * FROM Alumnos
```

Operaciones básicas de manipulación datos SQL. SELECT

Algunos ejemplos:

```
SELECT A.Nombre AS Nombre, A.Apellido AS Apellido
FROM Alumnos AS A
WHERE Clase = 'Desarrollo web'
```

Operaciones básicas de manipulación datos SQL. SELECT

Algunos ejemplos:

```
SELECT * FROM Alumnos WHERE Apellido LIKE '%Gar%'
```

Operaciones básicas de manipulación datos SQL. SELECT

Algunos ejemplos:

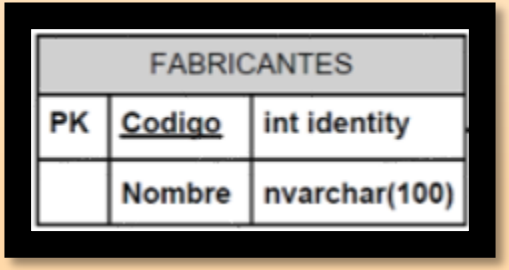
```
SELECT * FROM Alumnos
WHERE (Apellido = 'García' AND NotaMedia BETWEEN 5 AND 10)
```

Actividad 1

¿Cómo sería la instrucción para seleccionar el código de esta tabla?

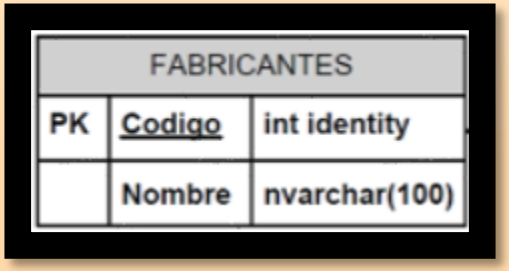
¿Y para solicitar todos los registros?

¿Y para ordenarlos de forma descendente según el nombre?



Actividad 2

¿Cómo sería la instrucción para añadir el resultado de una consulta a esta tabla?



FABRICANTES		
PK	Codigo	int identity
	Nombre	nvarchar(100)

Operaciones básicas de manipulación datos SQL. INSERT

- La instrucción INSERT de SQL permite añadir registros a una tabla.
- Podemos añadir uno a uno o varios de golpe.
- Veamos la sintaxis para cada uno de estos casos:

Operaciones básicas de manipulación datos SQL. INSERT

Insertando registros uno a uno

```
INSERT INTO Tabla [(Campo1, …, CampoN)] VALUES (Valor1, …, ValorN)
```

- **Tabla**: la tabla en la que se van a insertar las filas.
- **(Campo1, …, CampoN):** representa el campo o campos en los que vamos a introducir valores.
- **(Valor1, …, ValorN):** representan los valores que se van a almacenar en cada campo.

Operaciones básicas de manipulación datos SQL. INSERT

Insertando registros uno a uno

- Por ejemplo:

```
INSERT INTO Alumnos VALUES(1, 'Martin')
```

Operaciones básicas de manipulación datos SQL. INSERT

Inserción masiva de filas partiendo de consultas

- Se puede insertar de golpe múltiples registros en una tabla, cogiendo sus datos desde una consulta SELECT.

```
INSERT INTO Tabla [(Campo1, …, CampoN)]
SELECT ...
```

Operaciones básicas de manipulación datos SQL. INSERT

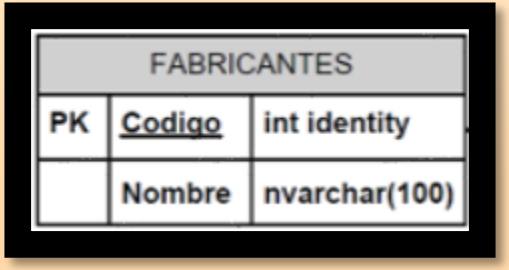
Inserción masiva de filas partiendo de consultas

- Por ejemplo:

```
INSERT INTO Alumnos
SELECT * FROM NuevosAlumnos WHERE Clase= 'Desarrollo web'
```

Actividad 3

¿Cómo sería la instrucción para añadir un registro a esta tabla?



Operaciones básicas de manipulación datos SQL. UPDATE

- Esta instrucción **nos permite actualizar los valores de los campos de una tabla**, para uno o varios registros, o incluso para todos los registros de una tabla.

Operaciones básicas de manipulación datos SQL. UPDATE

- Su sintaxis general es:

```
UPDATE Tabla
SET Campo1 = Valor1, ..., CampoN = ValorN
WHERE Condición
```

Operaciones básicas de manipulación datos SQL. UPDATE

- Siendo:
    - **Tabla:** Tabla en la que vamos a actualizar los datos.
    - **SET:** Campos que se van a actualizar y con qué valores lo vamos a hacer.
    - **WHERE:** Filtra qué registros de la tabla se van a actualizar.

Operaciones básicas de manipulación datos SQL. UPDATE

- Por ejemplo:

```
UPDATE Alumnos
SET Apellido = 'García'
WHERE Nombre = 'Martín'
```

Actividad 4

¿Cómo sería la consulta para actualizar el siguiente registro de la tabla?

Registro en la tabla: 01 – RAMÓN

Registro que lo sustituye: 01 – QUIQUE

FABRICANTES		
PK	Codigo	int identity
	Nombre	nvarchar(100)

Operaciones básicas de manipulación datos SQL. DELETE

- Esta instrucción **nos permite eliminar uno o múltiples registros. Incluso todos los registros de una tabla, dejándola vacía.**

Operaciones básicas de manipulación datos SQL. DELETE

- Su sintaxis general es:

```
DELETE [FROM] Tabla
WHERE Condición
```

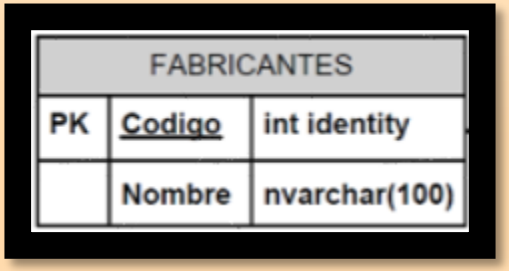
Operaciones básicas de manipulación datos SQL. DELETE

- Por ejemplo:

```
DELETE Alumnos WHERE DNI = '12345678Z'
```

Actividad 5

¿Cómo es la consulta para borrar todos los registros con el nombre “MARTÍN” ?



Operaciones básicas de manipulación datos SQL. DELETE

- Esta instrucción **nos permite eliminar uno o múltiples registros. Incluso todos los registros de una tabla, dejándola vacía.**

Acceso a base de datos. APIs

- Existen varias opciones disponibles para conectarse a un servidor MySQL desde una aplicación en PHP.
- PHP ofrece 3 APIs diferentes:
  - **mysql**
  - **mysqli**
  - **PDO**

Acceso a base de datos. APIs

- Tradicionalmente las conexiones entre PHP y MySQL se establecían utilizando la extensión nativa **mysql**
- Esta extensión se mantiene en la actualidad para dar soporte a las aplicaciones ya existentes que la utilizan, pero no se recomienda utilizarla
- Lo más habitual es **elegir entre mysqli (extensión nativa) y PDO.**

Acceso a base de datos. APIs

- Se podrán realizar acciones sobre las bases de datos como:
  - Establecer conexiones.
  - Ejecutar sentencias SQL.
  - Obtener los registros afectados o devueltos por una sentencia SQL.
  - Emplear transacciones.

Acceso a base de datos. Mysqli

- Desarrollada para aprovechar las nuevas funcionalidades encontradas en los sistemas MySQL con versión 4.1.3 o posterior.
- Cuenta con las siguientes mejoras con respeto a la extensión mysql:
  - Interfaz orientada a objetos
  - Soporte para consultas preparadas.
  - Soporte para transacciones.
  - Mejores opciones de depuración.

Acceso a base de datos. Mysqli. Configuración

- **En el fichero php.ini hay una sección específica para las opciones de configuración propias de cada extensión.**
- Entre las opciones que se pueden configurar para la extensión MySQLi están:
  - **mysqli.allow_persistent**: Permite crear conexiones persistentes.
  - **mysqli.default_port**: Número de puerto TCP predeterminado a utilizar cuándo se conecta al servidor de base de datos.

Acceso a base de datos. Mysqli. Configuración

- **mysqli.reconnect**: Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.
- **mysqli.default_host**: Host predeterminado a usar cuándo se conecta al servidor de base de datos.
- **mysqli.default_user**: Nombre de usuario predeterminado a usar cuándo se conecta al servidor de base de datos.
- **mysqli.default_pw**: Contraseña predeterminado a usar cuándo se conecta al servidor de base de datos.

Actividad 6

Encuentra el fichero php.ini y accede a la zona MySQLi para poder configurar las directivas

Acceso a base de datos. Mysqli. Conexión

- **Crear una instancia de la clase mysqli.**
    - **El constructor de la clase puede recibir seis parámetros**, todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:
        1. El nombre o dirección IP del servidor MySQL a lo que te quieres conectar.
        2. Un nombre de usuario con permisos para establecer la conexión.
        3. La contraseña del usuario.

Acceso a base de datos. Mysqli. Conexión

- **Crear una instancia de la clase mysqli.**
    - **El constructor de la clase puede recibir seis parámetros,** todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:

        4. El nombre de la base de datos a la que conectarse.
        5. El número del puerto en que se ejecuta el servidor MySQL.
        6. El socket a utilizar.

Acceso a base de datos. Mysqli. Conexión

- **Crear una instancia de la clase mysqli.**

```php
$db = new mysqli('localhost', 'usuario', 'contraseña', 'base_datos');
```

o

```php
$db = new  mysqli();
$db-> connect('localhost', 'usuario', 'contraseña', 'base_datos');
```

Actividad 7

- *Crea la BD **recetas** en MySQL*
- *Importa el script BDRecetas*
- *Conéctate usando una instancia de la clase mysqli*

Acceso a base de datos. Mysqli. Conexión

- **Obtener una instancia de la clase mysqli empleando la función mysqli_connect**

```
$db = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_datos');
```

También podemos establecer la conexión con el servidor y seleccionar a continuación la base de datos empleando la función mysqli_select_db:

```
$db= mysqli_connect('localhost', 'usuario', 'contraseña');
mysqli_select_db('base_datos',$db);
```

Acceso a base de datos. Mysqli. Conexión

- La function **mysqli_select_db** también la podemos usar para cambiar la base de datos con la que vamos a trabajar.
- Para esto también se puede usar el método select_db como se ve a continuación:

```
$db-> select_db("otra_bd");
```

Actividad 8

- *Crea la BD **EjerciciosMontecastelo** en MySQL*
- *Importa el script scriptEjerciciosMontecastelo*
- *Conéctate a la BD recetas y cambia a la BD EjerciciosMontecastelo utilizando los métodos necesarios*

Acceso a base de datos. Mysqli. Conexión

MUY IMPORTANTE:

- Verificar que la conexión se estableció correctamente antes de realizar ninguna operación sobre la base de datos.
- Se puede ver en las siguientes propiedades (o funciones equivalentes) de la clase mysqli:

Acceso a base de datos. Mysqli. Conexión

PROPIEDAD	FUNCIÓN	
connect_errno	mysqli_connect_errno( )	Devuelve el número de error generado por el último intento de conexión, o null si no se produce ningún error.
connect_error	mysqli_connect_error( )	Devuelve el mensaje de error generado por el último intento de conexión, o null si no se produce ningún error.

Acceso a base de datos. Mysqli. Conexión

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

```
$db = new mysqli('localhost', 'usuario', 'contraseña', 'Empresa');
$error = $db->connect_errno;

if ($error != null) {
    echo "<p>Error $error conectando a la base de datos:
$db->connect_error</p>";
    exit();
}
```

Actividad 9

Comprueba que la conexión a la BD se haya realizado correctamente

Acceso a base de datos. Mysqli. Conexión

CERRAR LA CONEXIÓN A LA BASE DE DATOS

- Para cerrar la conexión a base de datos podemos emplear la función **mysqli_close**:

```
mysqli_close($db);
```

- Empleando el estilo orientado a objetos podemos invocar el método **close()**:

```
$db->close();
```

Actividad 10

Cierra la conexión a las bases de datos una vez hayas acabado de trabajar con ellas

Acceso a base de datos. Mysqli. Errores

- La extensión MySQLi proporciona procedimientos y propiedades de objeto que dan información del último error producido en la base de datos:

PROPIEDAD	FUNCIÓN	
errno	mysqli_errno(mysqli $con)	Devuelve el número de error generado por el último intento de conexión, o null si no se produce ningún error.
error	mysqli_error(mysqli $con)	Devuelve el mensaje de error generado por el último intento de conexión, o null si no se produce ningún error.

Acceso a base de datos. Ejecución de consultas

- Al igual que cuando hicimos la conexión a la base de datos, podemos ejecutar consultas empleando mysqli de manera procedimental y orientado a objetos.
- Además, vamos a distinguir dos tipos de consultas:
    - **Las que devuelven valores**
    - **Las que no los devuelven**

Acceso a base de datos. Ejecución de consultas

Consultas que no devuelven valores

- La ejecución de consultas de creación y eliminación de bases de datos o tablas **(CREATE y DROP)**, y de actualización de la información de la base de datos **(UPDATE, INSERT y DELETE)** no devuelve ningún valor almacenado en la base de datos.
- Únicamente devolverá true o false para indicar si se ha producido algún error.

Acceso a base de datos. Ejecución de consultas

Consultas que no devuelven valores

- **La forma más inmediata de ejecutar una consulta con MySQLi es el método query**, utilizando el estilo orientado a objetos, o la **función mysqli_query** con el estilo procedimental.

Acceso a base de datos. Ejecución de consultas

Consultas que no devuelven valores

- **Ejemplo de consulta empleando el estilo orientado a objetos**

```php
$sentencia = "DROP TABLE IF EXISTS libro";
$db->query($sentencia);
if($db->errno)
{
    die('<br/>ERROR('.$db->errno.')->'.$db->error);
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que no devuelven valores

- **Ejemplo de consulta insertando una fila en la base de datos y comprobamos que el número de filas afectadas es 1**

```php
$sentencia = "INSERT INTO provincia (codigo, nombre)
              VALUES ('1', 'Madrid')";
$db->query($sentencia);
if($db->errno){
    die('< br/>ERROR('.$db->errno.')->'.$db->error);
}
if($db->affected_rows!=1){
    die('Error: no es el resultado esperado.');
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que no devuelven valores

- **Ejemplo de consulta utilizando el equivalente procedimental**

```
$sentencia = " INSERT INTO provincia (codigo, nombre)
              VALUES ('08', 'Barcelona')";
$resultado= mysqli_query($db, $sentencia);
if(mysqli_errno($db)){
    die('< br/>ERROR('.mysqli_errno($db).')->'.mysqli_error($db));
}
if(mysqli_affected_rows($db)!=1){
    die ('Error: no es el resultado esperado.');
}
```

Actividad 11

- *Inserta una receta utilizando el método procedimental*
- *Borra una receta utilizando la forma POO*

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **La ejecución de una sentencia SELECT sobre la base de datos devuelve un conjunto de resultados que habrá que procesar o mostrar.**
- Los datos se devuelven en forma de un objeto de la clase mysqli_result.
- Usamos los métodos:
    - **query** si usamos la interfaz orientada a objetos
    - función **mysqli_query** con el estilo procedimental.

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- Los métodos más comunes para manejar la información devuelta son:
  - **Array asociativo**
  - **Array asociativo, numérico o ambos**
  - **Objeto**

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **Array asociativo: array mysqli_result::fetch_assoc(void)**
- Este método maneja los datos de cada fila en un array asociativo, donde el nombre de cada columna es el índice. Habrá que crear un bucle para recurrir las filas y mostrar el resultado:

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **Por ejemplo:**

```php
$db = new mysqli("localhost", "alumno", "abcl23.", "recetas");
$sql = "SELECT chef.nombreartistico as chef, receta.nombre as receta FROM chef LEFT JOIN receta ON chef.codigo=cod_chef";
$resultado = $db->query($sql);
if ($db->errno) {
    die('< br/>ERROR(' . $db->errno . ')->' . $db->error);
}
while ($fila = $resultado->fetch_assoc()) {
    echo "CHEF:" . $fila['chef'] . "-->RECETA:" . $fila['receta'] . "<br/>";
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- El equivalente en estilo procedimental es el array asociativo **mysqli_fetch_assoc(mysqli_result $result).**

```php
$db = mysqli_connect("localhost", "alumno", "abc123.", "recetas");
$sql = "SELECT chef.nombreartistico as chef, receta.nombre as receta FROM chef LEFT JOIN receta ON chef.codigo=cod_chef";
$resultado = mysqli_query($db, $sql);
if (mysqli_errno($db)) {
    die('< br/>ERROR(' . mysqli_errno($db) . ')->' . mysqli_error($db));
}
while ($fila = mysqli_fetch_assoc($resultado)) {
    echo "CHEF:" . $fila['chef'] . "-->RECETA:" . $fila['receta'] . "<br/>";
}
```

Actividad 12

Selecciona todos los diferentes chefs y guárdalo en un array asociativo

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **Array asociativo, numérico o ambos:**

  mixed mysqli_result::fetch_array ([int $tiporesultado = MYSQLI_BOTH])

- Este método obtiene los datos de cada fila como un array asociativo, numérico o ambos dependiendo del parámetro que se le pase. Los posibles valores son:

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **MYSQLI_BOTH:**
  - Es el valor por defecto. Devuelve un array mixto de índices numéricos y claves (nombres de las columnas).

```
 Array (
[0] => MARTIN
[chef] => MARTIN
[1] => TORTILLA ESPAÑOLA
[receta] => TORTILLA ESPAÑOLA)
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **MYSQLI_ASSOC:**
  - Devuelve un array asociativo. Los nombres de las columnas serán las claves del array.

```
Array (
[chef] => MARTIN
[receta] => TORTILLA ESPAÑOLA)
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **MYSQLI_NUM:**
  - Devuelve un array numérico. Cada índice representa el contenido de las columnas de la consulta en la orden en la que están especificadas en esta.

```
Array (
[0] => MARTIN
[1] => TORTILLA ESPAÑOLA)
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- Para mostrar todas las columnas tendremos que hacer de manera similar al método anterior, indicando la clave en función del parámetro que se le pase:

```php
while ($fila = $resultado->fetch_array(MYSQLI_ASSOC)) {
    echo "CHEF:" . $fila['chef'] . "-->RECETA:" . $fila['receta'] . "<br/>";
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- El equivalente en estilo procedimental sería:

array mixed mysqli_fetch_array(mysqli_result $result [, int $tiporesultado = MYSQLI_BOTH])

```
while ($fila = mysqli_fetch_array($resultado, MYSQLI_ASSOC)) {
    echo "CHEF: " . $fila['chef'] . "->RECETA:" . $fila['receta'] . "<br/>";
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **Objeto**

  mysqli_result:: fetch_ object ([string $nombreclase [, array $params]])

- Este método **obtiene cada fila de resultados como si fuera un objeto**, $nombreclase es el nombre de la clase a instanciar (opcional) y $params es un array opcional de parámetros a pasar al constructor de dicha clase.

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- **Por ejemplo:**

```php
while ($objeto = $resultado->fetch_object()) {
    echo $objeto->chef . "-->" . $objeto->receta . "< br/>";
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- Usar este método nos permite instanciar una clase, la que le pasamos los datos de la fila del objeto, y que nos permite definir métodos para trabajar con estos.
- Los campos de la fila pasan a ser atributos públicos de la clase, con el cual podremos acceder a estos dentro de la clase sin necesidad de definirlos.

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- Por ejemplo, si tenemos una clase Consulta que recibe el resultado de nuestra consulta, automáticamente esta tendrá como atributos públicos chef y receta, con el cual podemos definir un método para mostrarlos como se ve a continuación:

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

```php
class Consulta {

    function mostrar() {
        return "< br/>" . $this->chef . "-->" . $this->receta;
    }

}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

- Para mostrar el contenido usamos el método **fetch_object** a lo que le pasamos la clase creada para recoger el resultado:

```
$resultado = $db->query($sql);
while ($fila = $resultado->fetch_object("Consulta")) {
    echo $fila->mostrar();
}
```

Acceso a base de datos. Ejecución de consultas

Consultas que devuelven valores

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

```
$resultado->free();
```

Actividad 13

Selecciona todas las recetas y guárdalas cada una en un objeto distinto

Acceso a base de datos. Transacciones

- **A veces queremos que X operaciones se ejecuten como un bloque**, esto es, que **o bien se ejecuten todas** correctamente, **o**, si alguna falta, no queremos que se registre **ningún cambio en la base de datos**.
- **Para esto debemos usar transacciones.**

Acceso a base de datos. Transacciones

- **Por defecto, MySQL se ejecuta en modo de ejecución automática** (autocommit), lo que significa que **cada consulta individual se incluye dentro de su propia transacción.**
- **Para poder revertir los cambios de las consultas (rollback)** usando transacciones, **debemos desactivar el modo de ejecución automática**, lo que iniciará la transacción

Acceso a base de datos. Transacciones

- Podemos deshabilitarlo usando la función:

```php
$db->autocommit(false);
```

- O con el estilo procedimental:

```php
mysqli_autocommit($db, false);
```

Acceso a base de datos. Transacciones

- **Para hacer efectiva las modificaciones:**
  - **Función commit**, con la interfaz orientada a objetos
  - **mysqli_commit(mysqli $link)**, con la interfaz procedimental.
- **Para revertir la transacción actual:**
  - **Función rollback**, con la interfaz orientada a objetos
  - **mysqli_rollback(mysqli $link)**, con la interfaz procedimental.

Acceso a base de datos. Transacciones

- A continuación, veremos un ejemplo completo de transacción con la interfaz orientada a objetos para insertar un nuevo chef del que no estaba dada de alta la provincia

Acceso a base de datos. Transacciones

```php
$db = new mysqli("localhost", "alumno", "abc123.", "recetas");
if ($db->connect_error) {
    echo "Error en la conexión a base de datos";
    exit;
} else {
    $bandera = true;
    $db->autocommit(FALSE);
    $sql1 = "INSERT INTO PROVINCIA (CODIGO,NOMBRE) VALUES('45','Toledo')";
    $sql2 = " INSERT INTO CHEF ( CODIGO, NOMBRE, APELLIDO1, SEXO,LOCALIDAD, COD_PROVINCIA) "
            ."VALUES (12,'MARTIN','GARCIA','M','Toledo','45')";
    $result = $db->query($sql1);
    if ($db->errno) {
        $bandera = false;
        echo '<br/>ERROR en la primera operación ('.$db->errno.')->'.$db->error;
    }
    $result = $db->query($sql2);
    if ($db->errno) {
        $bandera = false;
        echo '< br/>ERROR en la segunda operación ('.$db->errno.')->'.$db->error;
    }
    if ($bandera == true) {
        $db->commit();
        echo 'Transacción ejecutada con éxito!.';
    } else {
        $db->rollback();
        echo '< div>Error en algún punto de la transacción.div>';
    }
    $db->close();
}
```

Acceso a base de datos. Transacciones

- Veamos el mismo ejemplo con la interfaz procedimental:

Acceso a base de datos. Transacciones

```php
$db = mysqli_connect("localhost", "alumno", "abcl23.", "recetas");
if (mysqli_connect_error($db)) {
    echo "Error en la conexión a base de datos";
    exit;
} else {
    $bandera = true;
    mysqli_autocommit($db, FALSE);
    $sql1 = "INSERT INTO PROVINCIA (CODIGO, NOMBRE)  VALUES ('45','Toledo')";
    $sql2 = " INSERT INTO CHEF (CODIGO, NOMBRE, APELLIDO1, SEXO, LOCALIDAD,
     COD_PROVINCIA)  VALUES (12,'MARTIN','GARCIA',' M','Toledo','45')";
    $result = mysqli_query($db, $sql1);
    if (mysqli_errno($db) != 0) {
        $bandera = false;
        echo '<br/>ERROR en la primera operación ('
        .mysqli_errno($db).') ->' . mysqli_error($db);
    }
    $result = mysqli_query($db, $sql2);
    if (mysqli_errno($db) != 0) {
        $bandera = false;
        echo '< br/>ERROR en la segunda operación ('
        .mysqli_errno($db) .') ->' . mysqli_error($db);
    }
    if ($bandera == true) {
        mysqli_commit($db);
        echo 'Transacción ejecutada con éxito!.';
    } else {
        mysqli_rollback($db);
        echo '<div>Error en algún punto de la transacción. </div>';
    }
    mysqli_close($db);
}
```

Acceso a base de datos. Transacciones

- Es importante tener en cuenta algunas cosas cuando se trabaja con transacciones:
    - **El rollback únicamente afecta a las operaciones de manipulación de datos**, es decir, **DELETE, INSERT y UPDATE**. No afecta a CREATE, DROP o ALTER.

Acceso a base de datos. Transacciones

- **Si tenemos un campo auto_increment y durante la transacción se hace algún insert con éxito, y posteriormente falla alguna operación** realizada en la misma transacción, dando lugar a la ejecución de un rollback, **el campo auto_increment tendrá el contador como si los inserts sí hubieran tenido lugar.** El rollback no lo restaura al valor inicial.

- Para hacerlo **deberíamos ejecutar un ALTER TABLE** para establecer el valor a su estado actual.

Actividad 14

Prepara una inserción de un chef nuevo dentro de una transacción.

En caso de que falle, haz un rollback para que no se ejecute.

Acceso a base de datos. Consultas preparadas

- Cada vez que se envía una consulta al servidor, este debe analizarla antes de ejecutarla.
- Algunas sentencias SQL, como las que insertan valores en una tabla, deben repetirse de forma habitual en un programa.
- Para acelerar este proceso, **MySQL admite consultas preparadas.**

Acceso a base de datos. Consultas preparadas

- Estas consultas se almacenan en el servidor listas para ser ejecutadas cuando sea necesario y presentan las siguientes ventajas:
    - **Optimización**: Reducen el gasto de recursos en el análisis y ejecución de cada consulta.
    - **Seguridad**: Ofrecen más seguridad ante posibles inyecciones de SQL.

Acceso a base de datos. Consultas preparadas

- **Para trabajar con consultas preparadas con la extensión MySQLi** de PHP empleando la interfaz orientada a objetos, **se debe utilizar la clase mysqli_stmt.**
- **Utilizando el método stmt_init de la clase mysqli se obtiene un objeto de la dicha clase.**

Acceso a base de datos. Consultas preparadas

- Los pasos que se deben seguir para ejecutar una consulta preparada con la interfaz orientada a objetos son:
    - **Preparar la consulta** en el servidor MySQL **utilizando el método prepare.**
    - **Ejecutar la consulta**, tantas veces como sea necesario, **con el método execute**
    - **Cuando no se necesite más**, se debe **ejecutar el método close**.

Acceso a base de datos. Consultas preparadas

- Por ejemplo:

```
$consulta = $db-> stmt_init();
$consulta->prepare(' INSERT  INTO PROVINCIA ( codigo, nombre)
                     VALUES ("50", "Pontevedra")');
$consulta->execute();
$consulta->close();
$db->close();
```

Acceso a base de datos. Consultas preparadas

- De poco sirve preparar una consulta de inserción de datos si los valores que inserta son siempre los mismos.
- Por este motivo **las consultas preparadas admiten parámetros**.
- **Para preparar una consulta con parámetros**, **en vez de poner los valores debes indicar con un signo de interrogación su posición dentro de la sentencia SQL.**

Acceso a base de datos. Consultas preparadas

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

Acceso a base de datos. Consultas preparadas

- **Antes de ejecutar la consulta hay que utilizar el método bind_param para sustituir cada parámetro por su valor.**
- Siguiendo esta tabla, se define el carácter del primer parámetro:

CARACTER	TIPO DEL PARÁMETRO
I	Número entero
D	Número real (doble precisión)
S	Cadena de texto
B	Contenido en formato binario (BLOB)

Acceso a base de datos. Consultas preparadas

```php
$consulta = $db->stmt_init();
$consulta->prepare(' INSERT INTO PROVINCIA ( codigo, nombre)  VALUES (?, ?)');
$codigo = "50";
$provincia = "Pontevedra";
$consulta-> bind_param('ss', $codigo, $provincia);
$consulta->execute();
$consulta->close();
$db->close();
```

Acceso a base de datos. Consultas preparadas

- En el caso de las consultas que devuelven valores tenemos que **vincular las variables resultado de la consulta preparada para almacenar su resultado.**
- **Existen dos métodos** para extraer el resultado de las columnas de la consulta preparada:

Acceso a base de datos. Consultas preparadas

- **bind_result**: vincula las columnas del resultado de la columna con las variables que guardarán ese resultado.
- **fetch**: Permite obtener los resultados de esas variables, para lo cual deberemos recurrir un bucle que permita obtener los datos de todas las filas resultantes de la consulta.

Acceso a base de datos. Consultas preparadas

```
$consulta = $db->stmt_init();
$consulta->prepare(' SELECT nombre, dificultad, tiempo
      FROM receta  WHERE tiempo<50');
$consulta->execute();
$consulta->bind_result($receta, $dificultad, $tiempo);
while($consulta->fetch()) {
    print "<p>$receta($dificultad) - $tiempo minutos </p>";
}
$consulta->close();
$db->close();
```

Acceso a base de datos. Consultas preparadas

- Para emplear consultas preparadas con la extensión MySQLi de PHP empleando la interfaz procedimental **usaremos la función mysqli_stmt_init().**
- Los pasos a seguir son los mismos que se indicaron anteriormente, pero las funciones a emplear en este caso son: **mysqli_stmt_prepare**, **mysqli_stmt_execute** y **mysqli_stmt_close** respectivamente.

Acceso a base de datos. Consultas preparadas

```php
$sql = "INSERT  INTO PROVINCIA ( codigo, nombre)  VALUES (?, ?)";
$stmt = mysqli_stmt_init($db);
if( mysqli_stmt_prepare($stmt,$sql)){
```

Acceso a base de datos. Consultas preparadas

- La función para sustituir cada parámetro por su valor es **mysqli_stmt_bind_param**:

```
mysqli_stmt_bind_param($stmt, 'ss', $codigo, $provincia);
$codigo = "50";
$provincia = "Pontevedra";
mysqli_stmt_execute($stmt);
$codigo = "38";
$provincia = "Lugo";
mysqli_stmt_execute($stmt);
mysqli_stmt_close($stmt);
}
```

Acceso a base de datos. Consultas preparadas

- Para asignar a variables los campos que se obtienen tras la ejecución tenemos **mysqli_stmt_bind_result**
- Usamos la función **mysqli_stmt_fetch**() para recorrerlos

Acceso a base de datos. Consultas preparadas

```php
$db = mysqli_connect("localhost", "alumno", "abc123.", "recetas");
$sql = "SELECT nombre, dificultad, tiempo FROM receta  WHERE tiempo<50";
$stmt = mysqli_stmt_init($db);
if (mysqli_stmt_prepare($stmt, $sql)) {
    mysqli_stmt_execute($stmt);
    $receta = "";
    $dificultad = "";
    $tiempo = 0;
    mysqli_stmt_bind_result($stmt, $receta, $dificultad, $tiempo);
    while (mysqli_stmt_fetch($stmt)) {
        print "<p> $receta ($dificultad) -  $tiempo minutos </p>";
    }
    mysqli_stmt_close($stmt);
    $db->close();
}
```

Actividad 15

Utiliza una consulta preparada para insertar una nueva receta

Tema 7 :Acceso a base de datos MySQL

Ciclo Superior DAW

Asignatura: Desarrollo web en entorno servidor

Curso 20/21