DESARROLLO DE APLICACIONES WEB 2

ESTUDIO DE CLASES Y FUNCIONES PHP

15 DE NOVIEMBRE DE 2023

DESARROLLO WEB EN ENTORNO SERVIDOR
CARLOS GARCIA CACHON

CLASES

DOMDocument

Es una clase que permite trabajar con documentos XML permitiendo crear, cargar y manipular este tipo de archivos forma parte de DOM (Modelo de Objeto de Documento)

https://www.php.net/manual/en/class.domdocument.php

Propiedades

version

Indica la versión de XML que se utilizará en el documento.

```php
$dom = new DOMDocument('1.0');
echo $dom->version; // Imprime 1.0
```

encoding

Indica la codificación de caracteres utilizada en el documento.

```php
$dom = new DOMDocument('1.0', 'utf-8');
echo $dom->encoding; // Imprime utf-8
```

formatOutput

Formatea la salida

```php
//formatea la salida
$dom->formatOutput = true;
```

xmlStandalone

Controla si el documento XML debe considerarse independiente o no. Un documento XML independiente es aquel que no hace referencia a un documento DTD. Mediante true, false.

```php
//Indica con true que el documento no esta asoicado a un dtd
$dom->xmlStandalone = true;
```

resolveExternals

Controla si el analizador XML debe intentar resolver y cargar entidades externas, como archivos DTD o entidades externas definidas en el propio documento. Mediante true, false.

```php
//Indica si hay que resolver entidades externas o no como puede ser un dtd
$dom->resolveExternals = true;
```

documentElement

Representa el elemento raíz

```php
//Representa el elemento raiz del documento
$dom->documentElement;
```

validateOnParse

Valida un documento que tiene un dtd asociado

```php
$dom = new DOMDocument;
$dom->validateOnParse = true;
$dom->load('documento.xml');
```

Métodos

createElement()

Crea un nuevo elemento con el nombre especificado.

```php
$elemento = $dom->createElement('nombre_etiqueta', 'contenido_de_texto');
```

appendChild()

Agrega un hijo a un elemento

```php
$dom->appendChild($elemento);
```

createAttribute()

Crea un nuevo nodo de atributo con el nombre especificado.

```php
$atributo = $dom->createAttribute('nombre_atributo');
```

saveXML()

Devuelve una cadena que contiene la representación del documento en XML

```php
echo $dom->saveXML();
```

load()

Carga un documento XML desde una cadena o un archivo

```php
$dom->load('nombre_archivo.xml');
```

getElementsByTagName()

Devuelve una lista de elementos con un nombre de etiqueta específico

```php
$elementos = $dom->getElementsByTagName('nombre_etiqueta');
```

getAttribute()

Obtiene el valor de un atributo específico de un elemento

```
$valor = $elemento->getAttribute('nombre_atributo');
```

setAttribute()

Establece el valor de un atributo para un elemento.

```
$elemento->setAttribute('nombre_atributo', 'nuevo_valor');
```

Ejemplo de uso

En PHP

```
//Crea un fichero XML
$dom = new DOMDocument('1.0', 'utf-8');

// Crear elemento raíz
$raiz = $dom->createElement('raiz');
$dom->appendChild($raiz);

// Crear elemento hijo
$hijo = $dom->createElement('hijo', '¡Hola, XML!');
$raiz->appendChild($hijo);

// Generar salida XML
echo $dom->saveXML();
```

Esto genera

```
<?xml version="1.0" encoding="utf-8"?>
<raiz>
    <hijo>¡Hola, XML!</hijo>
</raiz>
```

Manejo de errores

Sirve para guardar los errores relacionados con el documento xml provocados durante la ejecución

```
$dom = new DOMDocument;
libxml_use_internal_errors(true);  // Habilitar el manejador de errores interno de libxml
$dom->load('documento_erroneo.xml');
$errores = libxml_get_errors(); // Guarda los errores ocurridos durante la ejecucion en $errores
libxml_clear_errors(); //limpia el listado de errores
```

Ismael Ferreras García

DATETIME

INTRODUCCION

Esta clase nos sirve para representar la fecha y la hora.

- **Documentación oficial** → https://www.php.net/manual/es/class.datetime.php
- **Versión PHP** → PHP 5 >= 5.2.0, PHP 7, PHP 8

SINOPSIS DE LA CLASE

Esta clase implementa la clase DateTimeInterfaces (https://www.php.net/manual/es/class.datetimeinterface.php)

Constantes más usadas

- **DateTime::ATOM** → Representa un formato de fecha y hora específico según el estándar ISO 860.

Corresponde al formato de fecha y hora en el siguiente formato:

- YYYY-MM-DDTHH:MM:SS±hh:mm.

  - 'YYYY' representa el año con cuatro dígitos.
  - MM representa el mes con dos dígitos.
  - DD representa el día con dos dígitos.
  - T es el separador entre la fecha y la hora.
  - HH representa la hora con dos dígitos (formato de 24 horas).
  - MM representa los minutos con dos dígitos.
  - SS representa los segundos con dos dígitos.
  - ±hh:mm representa la diferencia con la hora UTC en horas y minutos.

Por ejemplo, la fecha y hora actual en el formato DateTime::ATOM se vería algo así: 2023-11-18T15:30:00+00:00. Esta cadena representa el 18 de noviembre de 2023 a las 15:30:00 en el tiempo coordinado universal (UTC).

Puedes usar esta constante al trabajar con la clase DateTime en PHP para formatear fechas y horas según el estándar ISO 8601. Por ejemplo:

```php
$now = new DateTime();
$formattedDate = $now->format(DateTime::ATOM);
echo $formattedDate;
```

- **DateTime::COOKIE** → Representa un formato de fecha y hora, pero en este caso sigue el formato de fecha y hora definido por el estándar de cookies de HTTP.

Tiene la siguiente estructura:

- l, d-M-Y H: i:s T

  - 'l' representa el día de la semana en el idioma de la configuración regional.
  - 'd' representa el día del mes con dos dígitos.
  - 'M' representa el nombre corto del mes en el idioma de la configuración regional.
  - 'Y' representa el año con cuatro dígitos.
  - 'H' representa la hora en formato de 24 horas.
  - 'i' representa los minutos con dos dígitos.
  - 's' representa los segundos con dos dígitos.
  - 'T' representa la zona horaria abreviada.

Por ejemplo, la fecha y hora actual en el formato DateTime::COOKIE se vería algo así: "Saturday, 18-Nov-2023 15:30:00 UTC".

Puedes usar esta constante de la siguiente manera:

```php
$now = new DateTime();
$formattedDate = $now->format(DateTime::COOKIE);
echo $formattedDate;
```

Esto imprimirá la fecha y hora actual en el formato DateTime::COOKIE. Este formato es útil cuando necesitas mostrar fechas y horas en un formato fácilmente legible por humanos y compatible con las convenciones de las cookies HTTP.

Métodos más usados

- **DateTime::Constructor →** El constructor de la clase DateTime tiene varias formas de ser utilizado, pero la forma más común es la siguiente:
    - public DateTime::__construct([string $datetime = "now" [, DateTimeZone $timezone = NULL]])

Aquí hay una breve explicación de los parámetros:

- $datetime: Un string que representa la fecha y hora. Puede ser en varios formatos válidos de fecha y hora, como "now" para la fecha y hora actual, o en el formato "YYYY-MM-DD HH:MM:SS". Si este parámetro se omite, se asumirá "now" de manera predeterminada.
- $timezone: Un objeto DateTimeZone que representa la zona horaria en la que se debe interpretar la fecha y hora. Si se omite, se utilizará la zona horaria predeterminada del sistema.
    - Aquí hay algunos ejemplos de cómo puedes utilizar el constructor de la clase DateTime:

        - Crear un objeto DateTime con la fecha y hora actuales en la zona horaria predeterminada del sistema:

```php
$now = new DateTime();
```

        - Crear un objeto DateTime para una fecha y hora específicas:

```php
$customDate = new DateTime("2023-11-18 15:30:00");
```

- Especificar una zona horaria diferente:

```php
$customDateWithTimeZone = new DateTime("2023-11-18 15:30:00", new DateTimeZone("America/New_York"));
```

DateTime::add() → Este método agrega un objeto DateInterval al objeto DateTime. Un DateInterval representa un intervalo de tiempo, como una cantidad específica de días, horas, minutos, etc.

```php
public DateTime::add(DateInterval $interval)
```

- Aquí hay algunos ejemplos de cómo puedes utilizar este método

```php
$now = new DateTime("2023-11-18 15:30:00");
$interval = new DateInterval("P1D"); // Agrega un día
$now->add($interval);
echo $now->format("Y-m-d H:i:s"); // Mostrará la fecha y hora d
```

- En este ejemplo, se crea un objeto DateTime para el 18 de noviembre de 2023 a las 15:30:00. Luego, se crea un objeto DateInterval que representa un intervalo de tiempo de un día ("P1D"). Finalmente, el método add() se utiliza para agregar ese intervalo al objeto DateTime. El resultado se imprime con format() y mostrará la nueva fecha y hora después de agregar un día.

- Puedes ajustar el DateInterval según tus necesidades para agregar diferentes cantidades de días, horas, minutos, etc.

- **DateTime::setTimeZone()** → Establece la zona horaria para el objeto DateTime

```
public DateTime::setTimezone(DateTimeZone $timezone): DateTime
```

- Aquí hay una breve explicación de los parámetros:

  - Object → Solamente para el estilo por procedimientos: Un objeto DateTime devuelto por date_create(). La función modifica este objeto.

Timezone → Un objeto DateTimeZone que representa la zona horaria deseada.

- Devuelve el objeto DateTime para la cadena de métodos o false en caso de error.

- Aquí hay algunos ejemplos de uso

```php
<?php
$fecha = new DateTime('2000-01-01', new DateTimeZone('Pacific/Nauru'));
echo $fecha->format('Y-m-d H:i:sP') . "\n";

$fecha->setTimezone(new DateTimeZone('Pacific/Chatham'));
echo $fecha->format('Y-m-d H:i:sP') . "\n";
?>
```

- **DateTime::setTimeStamp()** → Establece la fecha y la hora basándose en una marca temporal de Unix

```php
public DateTime::setTimestamp(int $unixtimestamp): DateTime
```

- Aquí hay una breve explicación de los parámetros:

  - Object → Solamente para el estilo por procedimientos: Un objeto DateTime devuelto por date_create(). La función modifica este objeto.
  - Unixtimestamp→ La marca temporal de Unix que representa la fecha.

- Devuelve el objeto DateTime para la cadena de métodos o false en caso de error.
- Ejemplos de uso.

```php
<?php
$fecha = new DateTime();
echo $fecha->format('U = Y-m-d H:i:s') . "\n";

$fecha->setTimestamp(1171502725);
echo $fecha->format('U = Y-m-d H:i:s') . "\n";
?>
```

- **DateTime::format()** → Devuelve la fecha formateada según el formato dado

```php
public DateTime::format(string $format): string
```

- Aquí hay una breve explicación de los parámetros:

- Object → Solamente para el estilo por procedimientos: Un objeto DateTime devuelto por date_create()
- format→ Formato aceptado por date().

- Devuelve la fecha formateada en caso de éxito o false en caso de error.
- Ejemplo de uso

```php
<?php
$date = new DateTime('2000-01-01');
echo $date->format('Y-m-d H:i:s');
?>
```

- **DateTime::getLastErrors()** → Devuelve un array con las advertencias y los errores encontrados mientras se analizaba una cadena de fecha/hora.

```
public static DateTime::getLastErrors(): array|false
```

- Esta función no tiene parámetros.
- Devuelve un array que contiene información acerca de las advertencias y los errores, o false si no hay advertencias ni errores.

Ejemplo de uso

```php
<?php
try {
    $fecha = new DateTime('asdfasdf');
} catch (Exception $e) {
    // Sólo con propósitos de demostración...
    print_r(DateTime::getLastErrors());

    // La forma real orientada a objetos de hacer esto es
    // echo $e->getMessage();
}
?>
```

Álvaro Cordero Miñambres

PDO

- Funcionalidad

La funcionalidad de la clase PDO (**PHP Data Object**) es la proporción de una interfaz para interactuar con base de datos desde aplicaciones realizadas con el lenguaje de programación PHP.

- Constantes

Modos de error:

a. **PDO::ERRMODE_WARNING**: Configura PDO para que emita advertencias.

b. **PDO::ERRMODE_EXCEPTION**: Configura PDO para que lance excepciones.

Atributos de conexión:

a. **PDO::ATTR_ERRMODE**: Atributo para configurar el modo de error.

b. **PDO::ATTR_DEFAULT_FETCH_MODE**: Atributo para configurar el modo de obtención de resultados predeterminado.

Tipos de obtención de resultados:

a. **PDO::FETCH_ASSOC**: Devuelve un array indexado por nombres de columnas.

b. **PDO::FETCH_OBJ**: Devuelve un objeto anónimo con nombres de propiedades que corresponden a los nombres de las columnas.

Modos de ejecución de consultas:

a. **PDO::FETCH_CLASS**: Instancia un objeto y lo carga con datos de la fila.

Atributos de declaración:

a. **PDO::ATTR_CASE**: Atributo para configurar la forma en que se manejan los nombres de columnas en los resultados.

b. **PDO::ATTR_CURSOR**: Atributo para configurar el tipo de cursor.

c. **PDO::ATTR_TIMEOUT**: Atributo para configurar el tiempo de espera de la conexión.

Otros:

a. **PDO::PARAM_***: Varios atributos para establecer el tipo de parámetro en las consultas preparadas (por ejemplo, PDO::PARAM_INT, PDO::PARAM_STR).

- Auto-Commit

El término auto-commit hace referencia a la capacidad de asegurar todas las transacciones de una base de datos, siendo una transaccion un conjunto de operaciones (select, update, delete, insert), dichas estas se realizan en su totalidad o por el contrario no se realizará ninguna.

La propiedad del auto-commit de PDO trata de manera independiente cada una de sus operaciones y realiza su confirmación automáticamente, si alguna de estas es fallida se realizará un rollback o vuelta atrás para no realizar ninguna de las operaciones.

Ejemplo:

```php
// Establecer la conexión a la base de datos
$dsn = "mysql:host=localhost;dbname=nombre_base_de_datos";
$usuario = "nombre_usuario";
$contrasena = "contrasena";

try {
    $conexion = new PDO($dsn, $usuario, $contrasena);
} catch (PDOException $e) {
    echo "Error de conexión: " . $e->getMessage();
    exit;
}

// Desactivar autocommit y comenzar una transacción
$conexion->setAttribute(PDO::ATTR_AUTOCOMMIT, 0);
$conexion->beginTransaction();

try {
    // Realizar operaciones que forman parte de la transacción
    $conexion->exec("UPDATE tabla SET columna = 'nuevo_valor' WHERE id = 1'

    // Confirmar la transacción
    $conexion->commit();
} catch (PDOException $e) {
    // Revertir la transacción en caso de error
    $conexion->rollBack();
    echo "Error: " . $e->getMessage();
}

// Restaurar autocommit a su estado predeterminado
$conexion->setAttribute(PDO::ATTR_AUTOCOMMIT, 1);
```

- Sentacias preparadas

Las sentencias preparadas son una técnica en programación de bases de datos que se utiliza para mejorar el rendimiento al ejecutar consultas SQL.

En el contexto de PHP y la clase PDO, una sentencia preparada es una plantilla de consulta SQL que se precompila y se almacena en el servidor de la base de datos antes de ejecutarla, pudiendo así darle diferentes valores a los parámetros de la consulta y ejecutarla tantas veces fuera necesaria.

Ejemplos:

```php
// Establecer la conexión a la base de datos
$dsn = "mysql:host=localhost;dbname=nombre_base_de_datos";
$usuario = "nombre_usuario";
$contrasena = "contrasena";

try {
    $conexion = new PDO($dsn, $usuario, $contrasena);
} catch (PDOException $e) {
    echo "Error de conexión: " . $e->getMessage();
    exit;
}

// Sentencia preparada
$consulta = $conexion->prepare("INSERT INTO tabla (columna1, columna2)

// Asignar valores a los parámetros
$valor1 = "ejemplo1";
$valor2 = "ejemplo2";

$consulta->bindParam(':valor1', $valor1);
$consulta->bindParam(':valor2', $valor2);

// Ejecutar la sentencia preparada
$consulta->execute();

// Puedes reutilizar la sentencia preparada con diferentes valores
$valor1 = "nuevo_ejemplo1";
$valor2 = "nuevo_ejemplo2";

$consulta->execute();
```

- Metodos

Conexión a la base de datos:

a. **__construct($dsn, $username, $password, $options)**: Constructor que establece una conexión a la base de datos. $dsn es el Data Source Name, y $options son opciones de configuración.

Preparación y ejecución de consultas:

a. **prepare($statement, $driver_options)**: Prepara una sentencia SQL para ser ejecutada por el motor de la base de datos.
b. **exec($statement)**: Ejecuta una sentencia SQL y devuelve el número de filas afectadas.
c. **query($statement)**: Ejecuta una sentencia SQL y devuelve un conjunto de resultados como un objeto PDOStatement.

Manipulación de resultados:

a. **fetch($fetch_style, $cursor_orientation, $cursor_offset)**: Recupera la siguiente fila de un conjunto de resultados.
b. **fetchAll($fetch_style, $fetch_argument, $ctor_args)**: Devuelve un array que contiene todas las filas del conjunto de resultados.

Transacciones:

a. **beginTransaction()**: Inicia una transacción.
b. **commit()**: Confirma la transacción actual.
c. **rollBack()**: Revierte la transacción actual.

Gestión de errores:

a. **errorCode()**: Devuelve el código de error de la última operación de la base de datos.
b. **errorInfo()**: Devuelve información extendida sobre los errores ocurridos durante la última operación de la base de datos.

Configuración y ajustes:

a. **setAttribute($attribute, $value)**: Establece atributos de conexión.
b. **setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)**: Configura PDO para que lance excepciones en caso de errores.

Erika Martínez Pérez

PDOSTATEMENT

Funcionalidad y utilidad

- Sirve para el manejo de registros de una base de datos en un programa escrito en php
- Representa un conjunto de resultados (result set) de una consulta realizada a una base de datos
- Se genera al usar los métodos prepare(), query() o exec() de la clase PDO

Propiedades

- $queryString
  - ✓ Propiedad: publica
  - ✓ Tipo: String

Métodos

- PDOStatement::fetch
  - ✓ Obtiene una fila de un conjunto de resultados con un array y avanza el puntero por el conjunto de registros
  - ✓ Parámetros:
    - → $mode (opcional)
      - Controla cómo se devolverá la siguiente fila a la persona que llama
      - Tiene que tener el valor de alguna de estas constantes:
        - ✓ PDO::FETCH_OBJ(funciona igual que fetchObject())
        - ✓ PDO::FETCH_BOTH(predeterminado)
        - ✓ PDO::FETCH_ASSOC
        - ✓ PDO::FETCH_NAMED
        - ✓ PDO::FETCH_NUM
    - → $cursorOrientation (opcional)
      - Determina qué fila se devolverá del *result set*
      - Tipo: int
    - → $cursorOffset (opcional)
      - Especifica el número absoluto de la fila en el conjunto de resultados
      - Tipo: int
  - ✓ Returns: depende del valor de $mode, en el caso de ausencia, devuelve un array asociativo con los valores y los campos de la consulta. Devuelve *null* cuando termina de recorrer el conjunto de resultados.
  - ✓ Lanza una PDOException si el atributo *PDO::ATTR_ERRMODE* está establecido en *PDO::ERRMODE_EXCEPTION*
  - ✓ **Ejemplos:**
    - → Código con *PDO::FETCH_OBJ*

```php
<?php
$gsent = $gbd->prepare("SELECT name, colour FROM fruit");
$gsent->execute();
```

```php
print("PDO::FETCH_OBJ: ");
print("Devolver la siguiente fila como un objeto anónimo con nombres de columna como
propiedades\n");
$result = $gsent->fetch(PDO::FETCH_OBJ);
print $result->name;
print("\n");
?>
```

→ Resultado con *PDO::FETCH_OBJ*

```
PDO::FETCH_OBJ: Devolver la siguiente fila como un objeto anónimo con nombres de columna
como propiedades
kiwi
```

→ Código con *PDO::FETCH_BOTH*

```php
<?php
$gsent = $gbd->prepare("SELECT name, colour FROM fruit");
$gsent->execute();

print("PDO::FETCH_BOTH: ");
print("Devolver la siguiente fila como un array indexado por nombre y número de columna\n");
$result = $gsent->fetch(PDO::FETCH_BOTH);
print_r($result);
print("\n");
```

→ Resultado con *PDO::FETCH_BOTH*

```
PDO::FETCH_BOTH: Devolver la siguiente fila como un array indexado por nombre y número de
columna
Array
(
    [name] => banana
    [0] => banana
    [colour] => yellow
    [1] => yellow
)
```

- PDOStatement::fetchObject
  - ✓ Instancia un objeto PDOStatement que almacena un registro y avanza el puntero por el conjunto de registros
  - ✓ Parámetros:
    - → $class (opcional)
      - Nombre de la clase creada
      - Tipo: String
    - → $constructorArgs (opcional)
      - Los elementos de esta matriz se pasan al constructor
      - Tipo: array
  - ✓ Returns: Devuelve un objeto PDOStatement con propiedades que corresponden a los nombres de las columnas (Ejemplo: *$oUsuario->T01_Nombre*). Devuelve *false* en caso de error.
  - ✓ Emite un error con el nivel *E_WARNING* si el atributo *PDO::ATTR_ERRMODE* está establecido en *PDO::ERRMODE_WARNING*.
  - ✓ **Ejemplos:**
- PDOStatement::bindParam

- ✓ Bindea un parámetro al nombre de variable especificado, es decir, asigna un valor a un parámetro que aparece en la consulta SQL con una *?* o con 2 puntos seguido de una cadena de caracteres
- ✓ Parámetros:
  - → $parameter (obligatorio)
    - Si la consulta SQL está construida con ?, recibe el numero relativo a la posicion de la interrogación a la que queremos asignarle el valor ($variable)
    - Si la consulta SQL está construida con : seguido de una cadena, recibe la cadena con los 2 puntos a la que queremos asociarle el valor ($variable)
    - Tipo: String
  - → $variable (obligatorio)
    - Nombre de la variable que contiene el valor que le queremos asignar al parámetro ($parameter)
    - Tipo: variable php (cualquier tipo)
  - → $data_type (opcional)
    - El tipo de datos explícito para el parámetro usando las constantes *PDO::PARAM_** (parámetro por defecto: *PDO::PARAM_STR*)
    - Tipo: int
  - → $length (opcional)
    - Hace referencia a la longitud del tipo de datos
    - Tipo: int
- ✓ Returns: Devuelve *true* en caso de éxito o *false* en caso de error
- ✓ **Ejemplos:**
  - → Consulta con interrogaciones

```php
<?php
/* Ejecutar una sentencia preparada vinculando varialbes de PHP */
$calorías = 150;
$color = 'red';
$gsent = $gbd->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$gsent->bindParam(1, $calorías, PDO::PARAM_INT);
$gsent->bindParam(2, $color, PDO::PARAM_STR, 12);
$gsent->execute();
?>
```

  - → Consulta con :cadena

```php
<?php
/* Ejecutar una sentencia preparada vinculando varialbes de PHP */
$calorías = 150;
$color = 'red';
$gsent = $gbd->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$gsent->bindParam(':calories', $calorías, PDO::PARAM_INT);
$gsent->bindParam(':colour', $color, PDO::PARAM_STR, 12);
$gsent->execute();
?>
```

- PDOStatement::execute
  - ✓ Ejecuta una consulta preparada. Debe utilizarse después de PDO::prepare y de PDOStatement::bindParam para ejecutar la consulta con los parametros bindeados
  - ✓ Parámetros:
    - → $input_parameters (opcional)
      - Un array de valores con tantos elementos como parámetros vinculados en la sentencia SQL que va a ser ejecutada.
      - Tipo: array
  - ✓ Returns: Devuelve *true* en caso de éxito o *false* en caso de error
  - ✓ **Ejemplos:**
    - → Con parámetro

```php
<?php
/* Ejecutar una sentencia preparada proporcionando un array de valores de inserción */
$calorías = 150;
$color = 'red';
$gsent = $gbd->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$gsent->execute(array(':calories' => $calorías, ':colour' => $color));
?>
```

    - → Sin parámetro

```php
<?php
/* Ejecutar una sentencia preparada vinculando una variable y un valor */
$calorías = 150;
$color = 'gre';
$gsent = $gbd->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calorías AND colour LIKE :color');
$gsent->bindParam(':calories', $calorías, PDO::PARAM_INT);
$sth->bindValue(':colour', "%{$color}%");
$gsent->execute();
?>
```

- PDOStatement::rowCount
  - ✓ Almacena en una variable el número de filas afectadas por la última declaración SQL cuando ya ha sido preparada y ejecutada
  - ✓ Parámetros: No recibe parametros

- ✓ Returns: Devuelve un número entero asociado al número de filas afectadas por la consulta
- ✓ **Ejemplo:**
  - → Código

```php
<?php
/* Borrar todas las filas de la tabla FRUIT */
$del = $gbd->prepare('DELETE FROM fruit');
$del->execute();

/* Devolver el número de filas que fueron eliminadas */
print("Devolver el número de filas que fueron eliminadas:\n");
$cuenta = $del->rowCount();
print("Eliminadas $cuenta filas.\n");
?>
```

  - → Resultado

```
Devolver el número de filas que fueron eliminadas:
Eliminadas 9 filas.
```

- PDOStatement::fetchAll
  - ✓ Recupera las filas restantes de un conjunto de resultados
  - ✓ Parámetros:
    - → $fetch_style (opcional)
      - Controla el contenido del array devuelto al igual que *$mode* de PDOStatement::fetch (por defecto es *PDO::FETCH_BOTH*)
      - Tipo: int
    - → $fetch_argument (opcional)
      - Este argumento tiene un significado diferente dependiendo del valor del parámetro $fetch_style
      - Tipo: depende de $fetch_style
    - → $ctor_args (opcional)
      - Los argumentos del constructor de la case cuando el parámetro $fetch_style es *PDO::FETCH_CLASS*
      - Tipo: array
  - ✓ Returns: Devuelve un array que contiene todas las filas del conjunto de resultados
  - ✓ **Ejemplo:**
    - → Código

```php
<?php
$gsent = $gbd->prepare("SELECT name, colour FROM fruit");
$gsent->execute();

/* Obtener todas las filas restantes del conjunto de resultados */
print("Obtener todas las filas restantes del conjunto de resultados:\n");
$resultado = $gsent->fetchAll();
print_r($resultado);
?>
```

→ Resultado

```
Obtener todas las filas restantes del conjunto de resultados:
Array
(
    [0] => Array
        (
            [name] => pear
            [0] => pear
            [colour] => green
            [1] => green
        )

    [1] => Array
        (
            [name] => watermelon
            [0] => watermelon
            [colour] => pink
            [1] => pink
        )

)
```

Rebeca Sánchez Pérez

PDOEXCEPTION

- **Que es:** Representa un error generado por PDO. Las excepciones PDO son lanzadas cuando ocurren errores específicos al interactuar con la base de datos a través de PDO.

- **Sinopsis:**

```
class PDOException extends RuntimeException {

/* Propiedades */

public array $errorInfo;

protected string $code;


/* Propiedades heredadas */

protected string $message = "";

private string $string = "";

protected int $code;

protected string $file = "";

protected int $line;

private array $trace = [];

private ?Throwable $previous = null;


/* Métodos heredados */

final public Exception::getMessage(): string

final public Exception::getPrevious(): ?Throwable

final public Exception::getCode(): int

final public Exception::getFile(): string

final public Exception::getLine(): int

final public Exception::getTrace(): array

final public Exception::getTraceAsString(): string

public Exception::__toString(): string

private Exception::__clone(): void

}
```

- **Propiedades:**

  - ✓ PDO::errorInfo():PDO::errorInfo() devuelve un array con la información del error sobre la última operación realizada por el manejador de la base de datos. El array contiene los siguientes campos.

    public PDO::errorInfo(): array

  - ✓ Exception::getCode():Devuelve el código de Excepción en forma de int en Exception pero posiblemente en forma de otros tipos en Exception descendientes (por ejemplo como string en PDOException)

    final public Exception::getCode(): int

Borja Nuñez Refoyo

FUNCIONES

date()

json_encode()

Funcionalidad y Utilidad:

- Convierte una variable de PHP en su representación JSON.
- Enviar datos desde el servidor al cliente, por ejemplo, en una aplicación web
- Almacenar datos estructurados en archivos o bases de datos en formato JSON.

Parámetros Principales:

Tiene dos parámetros principales

- Datos para codificar ($data): Es el valor que se va a convertir a JSON. Puede ser un array, un objeto, una cadena de texto, números, booleanos y también admite valor NULL.
- Opciones ($options): Es un parámetro opcional que te permite especificar opciones para el proceso de codificación. Puedes usarlo para controlar el formato del JSON resultante. Algunas opciones comunes incluyen JSON_PRETTY_PRINT para formatear el JSON de manera legible y JSON_UNESCAPED_UNICODE para evitar el escapado de caracteres Unicode.

Ejemplo #1 Un ejemplo de json_encode()

```php
<?php
$data = array(
    "nombre" => "Juan",
    "edad" => 30,
    "ciudad" => "Ejemplo"
);

$json_data = json_encode($data);
```

Resultado:

```
{"nombre":"Juan","edad":30,"ciudad":"Ejemplo"}
```

Ejemplo #2 Un ejemplo de json_encode() + JSON_PRETTY_PRINT

```php
<?php
$data = array(
    "nombre" => "Juan",
    "edad" => 30,
    "ciudad" => "Ejemplo");
$json_data = json_encode($data, JSON_PRETTY_PRINT);
```

Resultado:

```
{
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "Ejemplo"
}
```

Ejemplo #3 Un ejemplo de json_encode() + JSON_UNESCAPED_UNICODE

```php
<?php
$data = array("nombre" => "Juan", "ciudad" => "東京");
$json_data = json_encode($data, JSON_UNESCAPED_UNICODE);
```

Resultado Con JSON_UNESCAPED_UNICODE:

```
{"nombre":"Juan","ciudad":"東京"};
```

Resultado Sin JSON_UNESCAPED_UNICODE:

```
{"nombre":"Juan","ciudad":"\u6771\u4EAC"}
```

Características Principales:

- UTF-8 por defecto: JSON usa UTF-8 como su formato de caracteres predeterminado, y json_encode() asume que los datos de entrada están en UTF-8.

Manejo de errores:

La función retorna false en caso de error durante la codificación. Puedes usar:

json_last_error() : Devuelve un código de error

json_last_error_msg() : Devuelve el mensaje de error correspondiente

```php
<?php
// JSON inválido
$json_invalido = '{"nombre":"Juan","edad":30,"ciudad":"Ejemplo",}'; // La coma del final hace saltar un error
$resultado = json_decode($json_invalido);

$last_error_code = json_last_error();
$last_error_msg = json_last_error_msg();

if ($last_error_code != JSON_ERROR_NONE) {
    echo 'Error al decodificar JSON. Código de error: ' . $last_error_code . '. Mensaje de error: ' . $last_error_msg;
} else {
    echo 'JSON decodificado correctamente.';
}
```

En este ejemplo, después de intentar decodificar el JSON inválido, se almacena el código de error en $last_error_code y el mensaje de error en $last_error_msg. Luego, se verifica si hay algún error ($last_error_code != JSON_ERROR_NONE) y, en caso afirmativo, se imprime tanto el código como el mensaje de error.

Resultado:

Error al decodificar JSON. Código de error: 4. Mensaje de error: Syntax error

Links:

https://www.php.net/manual/es/function.json-encode

json_decode()

Funcionalidad y Utilidad:

- Convierte una cadena JSON en una variable de PHP. Esto es especialmente útil cuando recibes datos en formato JSON, como en solicitudes HTTP o al leer archivos JSON.

Parámetros Principales:

Tiene dos parámetros principales:

- Cadena JSON ($json): Es la cadena que contiene la representación JSON que deseas decodificar.
- Asociatividad ($assoc, opcional): Este parámetro es opcional y determina si deseas que el resultado sea un array asociativo (true) o un objeto estándar (false). Por defecto, es false, lo que significa que el resultado será un objeto.

Ejemplo #1 Un ejemplo de json_decode()

```php
<?php
$json_data = '{"nombre": "Juan", "edad": 30, "ciudad": "Ejemplo"}';

// Decodificar la cadena JSON
$resultado = json_decode($json_data);

// Imprimir el resultado
var_dump($resultado);
```

Resultado:

```
object(stdClass)#1 (3) { ["nombre"]=> string(4) "Juan" ["edad"]=> int(30) ["ciudad"]=> string(7) "Ejemplo" }
```

Ejemplo #2 Un ejemplo de json_decode() + true

```php
<?php
$json_data = '{"nombre": "Juan", "edad": 30, "ciudad": "Ejemplo"}';

// Decodificar la cadena JSON
$resultado = json_decode($json_data,true);

// Imprimir el resultado
var_dump($resultado);
```

Resultado:

```
array(3) { ["nombre"]=> string(4) "Juan" ["edad"]=> int(30) ["ciudad"]=> string(7) "Ejemplo" }
```

Características Principales:

- Manejo de Objetos y Arrays: Puede generar tanto objetos como arrays dependiendo del valor de $assoc.
- Maneja tipos de datos comunes en JSON: Array, un objeto, una cadena de texto, números, booleanos y también admite valor NULL.

Manejo de errores:

Si hay algún error durante el proceso de decodificación devuelve NULL.

Puedes utilizar:

json_last_error() : Devuelve un código de error

json_last_error_msg() : Devuelve el mensaje de error correspondiente

```php
<?php
$json_invalido = '{"nombre": "Juan", "edad": 30, "ciudad": "Ejemplo",}'; // La coma del final hace saltar un error
$resultado_invalido = json_decode($json_invalido);

if ($resultado_invalido === null && json_last_error() !== JSON_ERROR_NONE) {
    echo 'Error al decodificar JSON. Código de error: ' . json_last_error() . '. Mensaje de error: ' . json_last_error_msg();
} else {
    var_dump($resultado_invalido);
}
```

En este ejemplo con una cadena JSON inválida, al intentar decodificar la con json_decode(), el resultado será NULL, indicando un fallo en la decodificación. Luego, se verifica si hay un error al comprobar si el código de error es diferente de JSON_ERROR_NONE, lo que indica la presencia de un error. En caso afirmativo, se imprime un mensaje detallado que incluye tanto el código como el mensaje de error,

Resultado:

Error al decodificar JSON. Código de error: 4. Mensaje de error: Syntax error

Links:

https://www.php.net/manual/es/function.json-decode

file_put_contents()

Funcionalidad y Utilidad:

- Sirve para escribir datos en un archivo. Su utilidad principal es simplificar el proceso de escritura de datos en un archivo, ya que realiza varias operaciones de archivo en una sola llamada.

Parámetros Principales:

Tiene dos parámetros principales:

- Nombre del Archivo ($filename): Es el nombre del archivo en el que deseas escribir los datos. Si el archivo no existe, se intentará crear. Si el archivo ya existe, los datos existentes se sobrescribirán, a menos que se especifique lo contrario.
- Datos ($data): Los datos que deseas escribir en el archivo. Esto puede ser una cadena de texto, un array o cualquier otro tipo de datos que pueda convertirse en una cadena.

  Ejemplo #1 Un ejemplo de file_put_contents()

```php
<?php
$filename = 'archivo.txt';
$data = 'Hola, mundo!';

// Escribir datos en el archivo
file_put_contents($filename, $data);
?>
```

  Resultado:

  En este ejemplo, la cadena "Hola, mundo!" se escribirá en el archivo llamado 'archivo.txt'. Si el archivo ya existe, su contenido se sobrescribirá.

- Flags ($flags, opcional): Este parámetro es opcional y se utiliza para especificar comportamientos adicionales.
  Algunos valores comunes son:
  - FILE_APPEND : Para agregar datos al final del archivo en lugar de sobrescribirlo.
  - LOCK_EX : Para adquirir un bloqueo exclusivo mientras se escribe en el archivo.

  Ejemplo #2 Un ejemplo de file_put_contents() + FILE_APPEND

```php
<?php
$filename = 'archivo.txt';
$data = 'Datos adicionales';

// Agregar datos al final del archivo
file_put_contents($filename, $data, FILE_APPEND);
```

  Resultado:

  En este ejemplo, si el archivo 'archivo.txt' ya contiene datos, la nueva cadena ("Datos adicionales") se agregará al final del archivo sin borrar el contenido existente. Si el archivo no existe, se creará y se escribirá la cadena en él.

Ejemplo #3 Un ejemplo de file_put_contents() + LOCK_EX

```php
<?php
$filename = 'archivo.txt';
$data = 'Nuevos datos';

// Escribir datos en el archivo con bloqueo exclusivo
file_put_contents($filename, $data, LOCK_EX);
```

Resultado:

En este ejemplo, se utiliza file_put_contents() con LOCK_EX para escribir los "Nuevos datos" en el archivo 'archivo.txt'. Durante este proceso, se adquiere un bloqueo exclusivo en el archivo, asegurando que ningún otro proceso pueda escribir en él simultáneamente.

Características Principales:

- Sobreescritura o Adición: Por defecto sobrescribe el contenido del archivo. Sin embargo, si usas la bandera FILE_APPEND, puedes agregar datos al final del archivo.
- Facilidad de Uso: file_put_contents() simplifica la tarea de escribir datos en un archivo en comparación con usar fopen(), fwrite() y fclose() de manera separada.
- Devuelve el número de bytes escritos en el archivo si la operación se realiza con éxito.

Manejo de Errores:

En caso de error retorna false si ocurre un error.

Puedes usar:

error_get_last() : devuelve un array asociativo que contiene información sobre el último error que ocurrió en el script. Este array tiene las siguientes claves:

type: El tipo de error.

message: El mensaje de error.

file: La ruta al archivo donde ocurrió el error.

line: El número de línea donde ocurrió el error.

```php
<?php
$filename = 'archivo_no_permisos.txt';
$data = 'Datos para escribir';

// Intentar escribir en un archivo sin permisos
$resultado = file_put_contents($filename, $data);

// Verificar si hay un error
if ($resultado === false) {
    $error = error_get_last();
    echo 'Error: ' . $error['message'] . ' en ' . $error['file'] . ' en la linea ' . $error['line'];
} else {
    echo 'Operación de escritura exitosa. Bytes escritos: ' . $resultado;
}
```

Resultado:

Error: file_put_contents(archivo_no_permisos.txt): failed to open stream: Permission denied en /home/EIRGo6/prog.php en la línea 10

Links:

https://www.php.net/manual/es/function.file-put-contents

header()

basename()

Recibe una URL