Servicio Nacional de Aprendizaje – SENA

Centro de Diseño y Metrologia

Informe php

Aprendiz:

Edwin Camilo Lozano Chaparro

No Ficha 2848530-A

}

Introducción:

En el desarrollo de aplicaciones web, el uso de bases de datos es fundamental para almacenar, consultar y manipular grandes cantidades de información de manera eficiente. **PHP** es uno de los lenguajes más populares para la creación de aplicaciones web dinámicas, y su extensión **MySQLi** permite interactuar con bases de datos MySQL de manera sencilla y eficaz. En este informe, se presenta una serie de ejemplos de código PHP que muestran cómo establecer una conexión a una base de datos MySQL, ejecutar consultas SQL y recuperar datos de una tabla. Además, se exploran las diferentes técnicas para manipular los resultados de estas consultas, como el uso de métodos como fetch_assoc(), fetch_object(), y el manejo adecuado de recursos con funciones como close(). El objetivo de este informe es proporcionar una comprensión básica de cómo interactuar con bases de datos en PHP utilizando la extensión MySQLi.

Contenido

Código 1

```
<html>
<body>
<?php
// El código php va aquí.
?>
</body>
</html>
```

1. **Estructura HTML**:
    - La etiqueta <html> indica el inicio del documento HTML.
    - <body> define el cuerpo del documento donde se colocará el contenido visible o procesado.
2. **Sección PHP**:
    - Entre las etiquetas <?php y ?> se puede escribir código PHP.
    - Actualmente, el bloque PHP está vacío, pero puede contener cualquier lógica de programación como operaciones, funciones o interacciones con un servidor.

Uso y propósito: Este es un ejemplo inicial para combinar HTML (para estructura visual) y PHP (para lógica y procesamiento). Se utiliza comúnmente en aplicaciones web dinámicas, como páginas que muestran datos procesados en el servidor.

Código 2

```php
<?php
// El código php va aquí.
// Cuando solamente hay código PHP se puede omitir el tag de
cierre.
```

1. **Apertura del bloque PHP**:
    - Se utiliza <?php para indicar el inicio de un bloque de código PHP.
2. **Comentario explicativo**:
    - El comentario explica que cuando un archivo contiene **únicamente código PHP**, no es necesario incluir el cierre ?>.
3. **Omisión del cierre**:
    - Esta práctica es recomendada en archivos con solo código PHP porque evita errores comunes, como enviar accidentalmente espacios o caracteres adicionales después del cierre, lo que puede causar problemas con las cabeceras HTTP.

Uso y propósito: Este enfoque se usa para escribir scripts PHP más seguros y limpios, especialmente en archivos dedicados al procesamiento del servidor, como controladores o configuraciones.

Código 3

```php
<?php
// este es comentario de una línea
# este es otro comentario de una línea.
echo "hola mundo" ;
```

```
?>
```

1. **Comentarios de una línea**:
    - // y # son formas de escribir comentarios de una línea en PHP.
    - Los comentarios son útiles para documentar el código o explicar su propósito, pero no afectan la ejecución del programa.
2. **Instrucción echo**:
    - echo "hola mundo"; imprime la cadena de texto **"hola mundo"** en la salida estándar (normalmente, en una página web o en la terminal).
    - El punto y coma (;) marca el final de la instrucción.

Salida esperada: Al ejecutar este script, se verá en pantalla:

hola mundo

Uso y propósito: Este código muestra cómo utilizar comentarios y realizar una operación básica (imprimir texto) en PHP. Es una base útil para aprender a desarrollar aplicaciones web dinámicas.

Código 4

```
<?php
/*
este es comentario abarca varias líneas.
Los comentarios son útiles para documentar los
programas.
```

```
*/
echo "hola mundo" ;
?>
```

1. **Comentarios de múltiples líneas**:
    - /* ... */ se utiliza para escribir comentarios que abarcan varias líneas.
    - Estos comentarios son ideales para documentar bloques grandes de código, explicar algoritmos o incluir notas extensas.
2. **Instrucción echo**:
    - echo "hola mundo"; imprime la cadena de texto **"hola mundo"** en la salida estándar.
    - El punto y coma (;) al final es obligatorio para finalizar la instrucción.

Salida esperada: Cuando se ejecuta este script, la salida será:

hola mundo

Uso y propósito: Este ejemplo muestra cómo incluir documentación clara dentro del código utilizando comentarios de múltiples líneas, además de realizar una operación básica como imprimir texto. Es una práctica esencial para mantener el código legible y entendible.

Código 5

```
<?php
// Ejemplos de declaración de variables.
$nombre = "Pedro"; // variable tipo texto;
```

```php
$apellido = "Perez"; // variable tipo texto;

$edad = 45; // variable numérica ;

echo "El señor ". $nombre . " " . $apellido . " tiene una

edad de ". $edad . " años. " ;

?>
```

1. **Declaración de variables**:
    - $nombre y $apellido son variables de tipo texto (string) que almacenan los valores "Pedro" y "Perez", respectivamente.
    - $edad es una variable numérica (integer) que almacena el valor 45.
2. **Concatenación de texto**:
    - Se utiliza el operador de punto (.) para concatenar cadenas de texto y valores de variables.
    - En la línea echo, se combinan cadenas estáticas con las variables para formar un mensaje dinámico.
3. **Instrucción echo**:
    - La función echo imprime el mensaje:

      "El señor Pedro Perez tiene una edad de 45 años."

Salida esperada: En la salida estándar (pantalla o terminal), se verá:

El señor Pedro Perez tiene una edad de 45 años.

Uso y propósito: Este código ilustra cómo declarar y usar variables en PHP, así como cómo crear mensajes dinámicos mediante concatenación. Es una práctica común en aplicaciones web dinámicas, como mostrar información personalizada para un usuario.

Código 6

```php
<?php
define( "<Nombre de la constante>", "<valor>" );
?>
```

1. **Uso de define()**:

- La función define() se utiliza en PHP para crear constantes.
- Una constante es un valor que no cambia durante la ejecución del programa.

2. **Sintaxis**:

- define("<Nombre de la constante>", "<valor>");
    - <Nombre de la constante>: Es el nombre de la constante (normalmente en mayúsculas por convención).
    - <valor>: Es el valor asignado a la constante, que puede ser un número, cadena de texto, etc.

Uso y propósito: Las constantes son útiles para almacenar valores que no cambian, como configuraciones del sistema, valores matemáticos o claves fijas.

Código 7

```php
<?php
$nombre = "Pedro";
$apellido = "Perez";
echo “Buenos días" . $nombre . " " . $apellido ;
?>
```

1. **Declaración de variables**:
    - $nombre y $apellido son variables que almacenan los valores de tipo texto "Pedro" y "Perez", respectivamente.
2. **Concatenación y salida**:
    - Se utiliza el operador de concatenación (.) para unir cadenas de texto con los valores de las variables.
    - La función echo imprime el saludo dinámico, combinando el texto fijo **"Buenos días"** con los valores de $nombre y $apellido.

Salida esperada (después de corregir): En la pantalla o terminal se mostrará:

Buenos días Pedro Perez

Uso y propósito: Este código demuestra cómo trabajar con variables, concatenar texto dinámico y generar una salida personalizada. Es útil en aplicaciones web donde se desea personalizar mensajes para los usuarios.

Código 8

```php
$arreglo = array(
"llave1" => "Valor1",
"llave2" => "Valor2",
);
```

1. **Declaración de un arreglo asociativo**:
    - $arreglo es un **arreglo asociativo** creado con la función array().
    - Un arreglo asociativo en PHP utiliza **claves** (o llaves) en lugar de índices numéricos para identificar los valores almacenados.
2. **Estructura del arreglo**:
    - "llave1" => "Valor1": La clave "llave1" está asociada al valor "Valor1".
    - "llave2" => "Valor2": La clave "llave2" está asociada al valor "Valor2".

Uso y propósito: Los arreglos asociativos son útiles para almacenar datos estructurados y facilitar el acceso mediante claves significativas, como en configuraciones, listas o registros.

Código 9

```php
<?php
$arreglo = [
"Nombre" => "Pedro",
"Apellido" => "Perez",
];
```

```
echo “Buenos días". $arreglo["Nombre"] . " " .
$arreglo["Apellido"] ;
?>
```

1. **Declaración de un arreglo asociativo:**
   - $arreglo es un **arreglo asociativo** definido con la sintaxis corta [], donde las claves son "Nombre" y "Apellido", y los valores asociados son "Pedro" y "Perez", respectivamente.
2. **Acceso a los valores del arreglo:**
   - Se accede a los valores dentro del arreglo usando las claves:
     - $arreglo["Nombre"] devuelve "Pedro".
     - $arreglo["Apellido"] devuelve "Perez".
3. **Concatenación y salida:**
   - La función echo se utiliza para imprimir un saludo, concatenando el texto **"Buenos días"** con los valores de las claves "Nombre" y "Apellido" del arreglo.

Código 10

```
<?php
if (expr)
Sentencia; // en caso de condición verdadera
else
Sentencia; // en caso de condición falsa
?>
```

1. **Estructura del condicional if-else:**
    - **if (expr)**: La estructura if evalúa la expresión expr. Si la condición es **verdadera** (es decir, si expr es un valor evaluado como verdadero), se ejecuta la **sentencia** que sigue al if.
    - **else**: Si la condición del if es **falsa** (es decir, si expr es evaluado como falso), se ejecuta la **sentencia** que sigue al else.
2. **Sintaxis:**
    - La expresión expr es una condición que puede ser cualquier comparación o lógica booleana. Si se cumple, se ejecuta el bloque del if; de lo contrario, se ejecuta el bloque del else.

Uso y propósito: La estructura if-else se utiliza para tomar decisiones en el flujo de ejecución del programa, permitiendo que el código se ejecute de manera diferente dependiendo de si una condición es verdadera o falsa. Es una de las estructuras de control más básicas y fundamentales en cualquier lenguaje de programación.

Código 11

```php
<?php
while (expr) //evalúa la condición
{
<instrucciones> ; //instrucción que se repite mientras
// condición sea verdadera
};
?>
```

1. **Estructura del ciclo while:**

- El ciclo while se utiliza para ejecutar un bloque de instrucciones repetidamente mientras se cumpla una **condición** especificada por la expresión expr.

2. **Funcionamiento:**

   - La condición expr se evalúa antes de ejecutar las instrucciones dentro del ciclo. Si la condición es **verdadera**, se ejecutan las instrucciones, y luego la condición se evalúa nuevamente. Este proceso continúa hasta que la condición sea **falsa**.
   - Si la condición es **falsa** desde el principio, el bloque de instrucciones no se ejecutará ni una sola vez.

Uso y propósito: El ciclo while es útil cuando se necesita repetir una acción o conjunto de acciones varias veces, pero no se conoce de antemano cuántas repeticiones serán necesarias. Se ejecuta hasta que la condición especificada sea falsa.

Código 12

```php
<?php
do
{
<instruccion 1> ;
<instruccion 2> ;
<instruccion n> ;
}
while (<expresión lógica>) // evalúa condición y repite ciclo en
//caso de ser verdadera
?>
```

1. **Estructura del ciclo do-while**:

    - El ciclo do-while es similar al ciclo while, pero con una diferencia clave: **las instrucciones dentro del ciclo se ejecutan al menos una vez**, independientemente de si la condición es verdadera o falsa.

2. **Funcionamiento**:

    - Primero, se ejecutan las instrucciones dentro del bloque do.
    - Después de ejecutar las instrucciones, se evalúa la **expresión lógica** en el while. Si la condición es **verdadera**, el ciclo se repite. Si es **falsa**, el ciclo termina.
    - Este ciclo garantiza que el bloque de instrucciones se ejecute al menos una vez, ya que la condición se evalúa **después** de la ejecución.

Uso y propósito: El ciclo do-while es útil cuando es necesario ejecutar un bloque de instrucciones al menos una vez antes de evaluar la condición, como en formularios de entrada donde se requiere que el usuario ingrese datos al menos una vez antes de hacer una validación.

Código 13

```
<?php
for (expresión 1; expresión 2; expresión 3)
{
<sentencia 1> ;
<sentencia 2> ;
<sentencia n> ;
```

```
} ;
?>
```

1. **Estructura del ciclo for**:
    - El ciclo for en PHP se utiliza para ejecutar un bloque de código un número determinado de veces, según se define en las tres expresiones.
2. **Funcionamiento**:
    - **Expresión 1**: Se ejecuta una sola vez al inicio del ciclo. Generalmente se usa para inicializar una variable de control (como $i = 0).
    - **Expresión 2**: Es la condición que se evalúa antes de cada iteración. Mientras sea **verdadera**, el ciclo continuará ejecutándose.
    - **Expresión 3**: Se ejecuta al final de cada iteración, generalmente para actualizar la variable de control (como $i++ para incrementar el valor de $i).

Uso y propósito: El ciclo for es útil cuando se conoce de antemano el número de veces que se debe ejecutar un bloque de código, como en la iteración de arreglos o la realización de tareas repetitivas con un contador.

Código 14

```
<?php
switch (variable_a_evaluar)
```

```
{
case <valor1>:
<sentencias> ;
break;
case <valor2>:
<sentencias> ;
break;
case <valorn:
<sentencias> ;
break;
default // si no corresponde con ninguno de los
// valores anteriores
<sentencias> ;
}
?>
```

1. **Estructura del switch**:
    - El switch es una estructura de control de flujo que evalúa una variable o expresión contra varios valores posibles y ejecuta el bloque de código correspondiente al valor que coincida.
2. **Funcionamiento**:
    - **switch (variable_a_evaluar)**: Evalúa la **variable** o **expresión** proporcionada.
    - **case <valor>**: Si la **variable_a_evaluar** coincide con el valor de un case, se ejecutan las sentencias asociadas a ese caso.

- **break;**: Sale del switch después de ejecutar el bloque de sentencias del caso correspondiente. Si se omite, el flujo continúa ejecutando los siguientes casos, incluso si no coinciden (lo que se llama "fall-through").
- **default:**: Es un bloque opcional que se ejecuta si ninguno de los valores case coincide con la variable evaluada.

Uso y propósito: El switch es útil cuando se tienen varias opciones posibles y se desea evitar múltiples sentencias if-else. Es más legible y eficiente cuando se evalúan múltiples valores posibles para una variable.

Código 15

```
<?php
foreach (<nombre_del_arreglo> as <variable_auxiliar>)
{
<sentencia 1> ;
<sentencia 2> ;
<sentencia n> ;
} ;
?>
```

1.Estructura del ciclo foreach:

- El ciclo foreach es utilizado para iterar sobre **arreglos** o **colecciones** en PHP. A diferencia de otros ciclos, no es necesario especificar el índice del arreglo, ya que se recorre automáticamente cada elemento

2.Funcionamiento:

- **<nombre_del_arreglo>**: Es el arreglo que se va a recorrer.
- **<variable_auxiliar>**: Es una variable temporal que tomará el valor de cada elemento del arreglo en cada iteración.
- El ciclo ejecuta las instrucciones dentro de su bloque para cada elemento del arreglo. Esto es útil para trabajar con todos los elementos de un arreglo sin necesidad de gestionar índices manualmente.

Código 16

```php
<?php
$arreglo = [ "Jacinto", "Jose", "Pepita", "Mendieta" ] ;
$j = 0;
foreach($arreglo as $elemento)
{
echo "$j: $elemento \n";
++$j ;
}
?>
```

1. **Definición del arreglo**:

   - Se crea un arreglo $arreglo con los valores "Jacinto", "Jose", "Pepita", y "Mendieta".

2. **Inicialización de una variable**:

   - La variable $j se inicializa en 0. Esta variable se usará para mostrar un contador junto con los elementos del arreglo.

3. **Ciclo foreach**:

   - Se utiliza el ciclo foreach para recorrer todos los elementos del arreglo $arreglo.
   - En cada iteración, la variable $elemento toma el valor de cada elemento del arreglo, es decir, "Jacinto", "Jose", "Pepita", y "Mendieta" en sucesivas iteraciones.
   - **Impresión de los resultados**: Dentro del ciclo, se utiliza echo para imprimir el índice (almacenado en $j) y el valor del elemento actual del arreglo ($elemento). Después de imprimir, la variable $j se incrementa con ++$j.

Uso y propósito: Este código es útil para recorrer arreglos y asociar un índice con cada elemento durante la iteración, permitiendo que se impriman tanto los índices como los valores de los elementos del arreglo.

Código 17

```
<?php
function(<parámetros>){
// bloque de código
```

```
return <variable>
};
?>
```

1. Definición de una función en PHP:

- En PHP, una **función** se define utilizando la palabra clave function, seguida de un nombre para la función y los **parámetros** que la función recibirá.

2. Partes de la función:

- **function nombre_de_funcion**: Define el nombre de la función. Este nombre debe seguir las reglas de nomenclatura de PHP y ser único en el contexto donde se usa.
- **<parámetros>**: Son los valores que se pasan a la función cuando se llama. Los parámetros pueden ser variables o valores específicos y pueden ser opcionales.
- **Bloque de código**: Dentro de las llaves { } va el código que la función ejecutará cuando se llame.
- **return <variable>**: El return devuelve un valor desde la función a quien la haya llamado. La función puede retornar un valor, que puede ser una variable, una operación o cualquier tipo de dato.

Uso y propósito: Las funciones en PHP permiten encapsular bloques de código reutilizables. Son útiles para modularizar el código, mejorar su legibilidad y evitar la repetición de procesos.

Código 18

```
<?php
```

```
include_once ’’<nombre_biblioteca.php>’’ ;
?>
```

1. Incluir una biblioteca o archivo PHP:

- El código include_once se utiliza para **incluir** un archivo externo en el script PHP. Esto permite reutilizar código que se encuentra en otro archivo, como funciones, clases o configuraciones, sin necesidad de duplicar el código.

2. Funcionamiento de include_once:

- **include_once**: Esta instrucción incluye el archivo especificado, pero **solo una vez** durante la ejecución del script. Si el archivo ya ha sido incluido previamente, no se incluirá de nuevo, lo que previene errores de definición repetida o duplicada.
- Es útil cuando quieres asegurarte de que el archivo sea incluido una sola vez, incluso si el código de inclusión está en varias partes del programa.

Uso y propósito: include_once es útil cuando deseas dividir el código en varios archivos y mantener la modularidad, evitando incluir el mismo archivo múltiples veces. También asegura que el archivo se cargue solo una vez, lo que es importante para evitar errores de redefinición de funciones o clases.

Código 19

```
<a href=”holamundo.php?nombre=pedro&apellido=perez> Enlace a
Pedro Pérez </a>
```

1. Enlace con parámetros GET en PHP:

- El código proporcionado es un ejemplo de un **enlace HTML** que utiliza el método **GET** para enviar parámetros a un archivo PHP.
- **<a>**: La etiqueta <a> se utiliza para crear un enlace o hipervínculo en HTML.

- **href="holamundo.php?nombre=pedro&apellido=perez"**: El atributo href define la dirección URL a la que se redirige al hacer clic en el enlace. En este caso, la URL contiene parámetros que se pasan al archivo holamundo.php.
    - **Parámetros GET**: ?nombre=pedro&apellido=perez son parámetros de la URL. El parámetro nombre tiene el valor "pedro", y el parámetro apellido tiene el valor "perez". Estos valores serán enviados a holamundo.php cuando el usuario haga clic en el enlace.

Uso y propósito: Este tipo de enlace es útil para pasar datos desde un formulario o una página a otra. El método GET es común para pasar datos a través de la URL en una solicitud, lo que permite compartir o procesar información entre diferentes páginas.

Código 20

```
<form action=”<nombre_programa>.php”>
Enunciado del campo:
<input type=”text” name=”campo1”>
<input type=”submit” value=”Enviar”>
</form>
```

1. Formulario HTML con método POST o GET:

- El código proporcionado crea un **formulario HTML** que permite al usuario ingresar datos y enviarlos a un programa PHP para su procesamiento. En este caso, el formulario utiliza el método **GET** (por defecto) para enviar los datos, aunque se puede modificar para usar **POST**.
- **<form action="nombre_programa.php">**: La etiqueta <form> crea el formulario, y el atributo action define el archivo PHP al que se enviarán los datos del formulario. En este caso, los datos se enviarán a <nombre_programa>.php cuando se presione el botón de envío.
- **<input type="text" name="campo1">**: Crea un campo de texto donde el usuario puede ingresar información. El atributo name especifica el nombre del campo (en este caso, campo1). Este nombre será usado en el archivo PHP para acceder al valor ingresado por el usuario.
- **<input type="submit" value="Enviar">**: Crea un botón de envío que envía el formulario al archivo PHP cuando se hace clic en él. El atributo value especifica el texto que aparece en el botón ("Enviar").

2. Recibiendo los datos en PHP:

- Los datos enviados desde el formulario se pueden recuperar en el archivo PHP usando las superglobales $_GET o $_POST (dependiendo del método elegido en el formulario).
- Si el formulario utiliza **GET** (por defecto) para enviar los datos, se accedería al campo campo1

Uso y propósito: El formulario HTML con el atributo action es fundamental para enviar datos desde el cliente (usuario) hacia el servidor, donde serán procesados por un archivo PHP. Se utiliza para recolectar entradas de los usuarios, como texto, selecciones, archivos, etc.

Código 21

```
create database adsi ;

create table ciudades ( codigo text, nombre text ) ;

insert into ciudades ( codigo, nombre ) values ( "05001", "MEDELLIN" );

insert into ciudades ( codigo, nombre ) values ( "05002", "ABEJORRAL" );

insert into ciudades ( codigo, nombre ) values ( "05004", "ABRIAQUI" );

insert into ciudades ( codigo, nombre ) values ( "05021", "ALEJANDRIA" );
```

El código proporcionado está relacionado con **SQL** (Structured Query Language), y se utiliza para crear una base de datos y una tabla, así como para insertar registros en ella. A continuación, se explica cada parte del código:

1. Crear la base de datos:

- **create database adsi;**: Esta instrucción crea una nueva base de datos llamada adsi. Una base de datos es un contenedor para almacenar tablas y otros objetos relacionados en un sistema de gestión de bases de datos (DBMS).

2. Crear la tabla:

- **create table ciudades**: Crea una nueva tabla llamada ciudades en la base de datos adsi.
- **codigo text, nombre text**: Define las columnas de la tabla:

- **codigo**: Una columna de tipo text que almacenará códigos (en este caso, los códigos de las ciudades).
- **nombre**: Una columna de tipo text que almacenará los nombres de las ciudades.

3. Insertar registros en la tabla:

Las instrucciones insert into añaden datos a la tabla ciudades:

- **insert into ciudades (codigo, nombre) values ("05001", "MEDELLIN");**: Inserta el código 05001 y el nombre de la ciudad MEDELLIN.
- Se repiten las mismas instrucciones para insertar más ciudades: ABEJORRAL, ABRIAQUI y ALEJANDRIA con sus respectivos códigos.

Uso y propósito: Este código se utiliza para crear una base de datos simple que almacena información sobre ciudades. En este caso, se almacena un código único para cada ciudad y su nombre. La base de datos y las tablas son fundamentales para organizar y acceder a los datos de manera estructurada en un sistema de gestión de bases de datos.

Código 22

```
$conn = new mysqli("<host>", "<usuario>", "<clave>", "<base_de_datos>" ) ;
 if( $conn->connect_errno ) {
 echo "Falla al conectarse a Mysql ( ". $conn->connect_errno . ") " .
 $conn->connect_error ;
 } else {
 echo $conn->host_info. "\n" ;
} ;
```

Este código está escrito en **PHP** y utiliza la **extensión MySQLi** para conectarse a una base de datos MySQL. A continuación se describe cada parte del código:

1. Conexión a la base de datos MySQL:

new mysqli(...): Crea una nueva instancia de la clase mysqli, que se utiliza para interactuar con una base de datos MySQL en PHP.

Los parámetros proporcionados dentro de mysqli() son:

- **<host>**: El servidor MySQL al que se conecta. Puede ser localhost si el servidor está en la misma máquina, o una dirección IP si está en un servidor remoto.
- **<usuario>**: El nombre de usuario para autenticar la conexión a la base de datos.
- **<clave>**: La contraseña asociada al nombre de usuario para acceder a la base de datos.
- **<base_de_datos>**: El nombre de la base de datos a la que se quiere conectar.

2. Verificación de la conexión:

- **$conn->connect_errno**: Verifica si ha ocurrido algún error en la conexión. Si la conexión falla, esta propiedad contendrá el código del error.
- **$conn->connect_error**: Devuelve el mensaje de error si la conexión no es exitosa.
- Si hay un error en la conexión, se imprime un mensaje detallado con el código de error y la descripción.

3. Éxito en la conexión:

- Si la conexión es exitosa, se ejecuta el bloque else. En este caso, se imprime información sobre el servidor MySQL al que se está conectado usando $conn->host_info. Este atributo devuelve

una cadena con detalles sobre el host de la base de datos, como el tipo de conexión y la versión del servidor.

Uso y propósito: Este código es comúnmente utilizado para conectar un script PHP con una base de datos MySQL, lo que permite ejecutar consultas SQL, obtener datos y realizar operaciones de inserción, actualización o eliminación en la base de datos. La verificación de errores es importante para manejar cualquier problema de conexión y evitar fallos inesperados en la aplicación.

Código 23

```php
<?php
$conn = new mysqli("localhost", "desarrollador", "adsi2017", "citas" ) ;
if( $conn->connect_errno) {
echo "Falla al conectarse a Mysql ( ". $conn->connect_errno . ") " .
$conn->connect_error ;
} else {
echo $conn->host_info. "\n" ;
} ;
?>
```

Este código está escrito en **PHP** y utiliza la **extensión MySQLi** para conectarse a una base de datos MySQL. A continuación se explica cada parte del código:

1. Establecer la conexión con la base de datos MySQL:

Los parámetros proporcionados en el constructor mysqli() son:

- **"localhost"**: El nombre del servidor MySQL al que se va a conectar. En este caso, localhost indica que el servidor está en la misma máquina donde se ejecuta el script PHP.
- **"desarrollador"**: El nombre de usuario utilizado para autenticar la conexión.
- **"adsi2017"**: La contraseña asociada al nombre de usuario desarrollador.
- **"citas"**: El nombre de la base de datos a la que se está conectando.

2. Verificación de la conexión:

- **$conn->connect_errno**: Si ocurre un error en la conexión, esta propiedad contiene un código de error numérico. Si no hay error, será 0.
- **$conn->connect_error**: Contiene una cadena con el mensaje de error proporcionado por MySQL si la conexión falla.
- Si ocurre un error de conexión, el mensaje de error será mostrado, incluyendo el código y la descripción del error.

3. Conexión exitosa:

- Si la conexión es exitosa, el bloque else se ejecuta, y se imprime información sobre el servidor MySQL al que se ha conectado. Esta información está disponible a través de $conn->host_info, que devuelve detalles como el tipo de conexión y la versión del servidor.

Uso y propósito: Este código establece una conexión con una base de datos MySQL utilizando las credenciales proporcionadas (localhost, desarrollador, adsi2017, citas). La verificación de errores es fundamental para asegurar que la conexión se haya realizado correctamente y poder manejar adecuadamente cualquier problema que surja, como contraseñas incorrectas o problemas con el servidor.

Código 24

```
if( $mysqli->query(“<sentencia sql>”) === TRUE ) {
   echo “Se ejecutó la sentencia con éxito”;
   } else {
   echo “Hubo un error ...” ;
   };
```

Este fragmento de código está escrito en **PHP** y utiliza la extensión **MySQLi** para ejecutar una consulta SQL en una base de datos MySQL. El código también maneja los resultados de la ejecución de la consulta. A continuación, se explica cada parte del código:

1. Ejecutar una consulta SQL:

$mysqli->query("<sentencia sql>"): Esta instrucción ejecuta una consulta SQL en la base de datos utilizando el objeto $mysqli (que debe haber sido previamente inicializado como una conexión a MySQLi). El parámetro **"<sentencia sql>"** debe ser reemplazado por una consulta SQL válida, como una instrucción SELECT, INSERT, UPDATE, DELETE, etc.

- Si la consulta se ejecuta correctamente y no devuelve un resultado (como en el caso de una sentencia INSERT, UPDATE, DELETE), la función query() retorna **TRUE**.
- Si ocurre algún error o la consulta es incorrecta, query() retornará **FALSE**.

2. Comprobación de si la consulta fue exitosa:

- Si la consulta SQL se ejecutó correctamente (es decir, la función query() devolvió TRUE), se imprime el mensaje **"Se ejecutó la sentencia con éxito"**.

3. Manejo de errores en caso de que la consulta falle:

- Si la consulta no se ejecutó correctamente (es decir, si la función query() devolvió FALSE), el código dentro del bloque else se ejecuta, mostrando el mensaje **"Hubo un error ..."**. En este caso, podrías agregar detalles adicionales sobre el error utilizando el método $mysqli->error para obtener una descripción más detallada del problema

Uso y propósito: Este código se utiliza para ejecutar una consulta SQL y manejar su éxito o error. Es común en aplicaciones web que interactúan con bases de datos, permitiendo realizar operaciones de lectura y escritura (como insertar, actualizar o eliminar datos) y gestionando posibles fallos.

Código 25

```
if( $mysqli->query(“<sentencia sql>”) === TRUE ) {
   echo “Se ejecutó la sentencia con éxito”;
   } else {
   echo “Hubo un error ...” ;
   };
```

Este fragmento de código está escrito en **PHP** y utiliza la extensión **MySQLi** para ejecutar una consulta SQL en una base de datos MySQL y procesar los resultados. A continuación se explica cada parte del código:

1. Ejecutar la consulta SQL y obtener el resultado:

- **$mysqli->query("<sentencia sql>")**: Ejecuta la consulta SQL definida por **"<sentencia sql>"** (como un SELECT, por ejemplo) en la base de datos utilizando el objeto $mysqli (que debe ser previamente inicializado como una conexión a MySQLi).
- Si la consulta SQL se ejecuta correctamente y devuelve un conjunto de resultados (como una consulta SELECT), el resultado de la ejecución se almacena en la variable $resultado.
- Si la consulta falla (por ejemplo, por errores en la sintaxis), el bloque else se ejecutará.

2. Obtener el número de registros devueltos:

- **$resultado->num_rows**: Después de ejecutar una consulta SELECT, se puede utilizar la propiedad num_rows del objeto $resultado para obtener el número de filas que fueron devueltas por la consulta.
- En este caso, se imprime un mensaje que indica cuántos registros fueron devueltos por la consulta. Por ejemplo, si la consulta devuelve 5 registros

3. Manejo de errores en caso de que la consulta falle:

- Si la consulta no se ejecuta correctamente (por ejemplo, debido a un error de sintaxis SQL), el bloque else se ejecuta y se muestra el mensaje **"Hubo un error ..."**. Para obtener más detalles sobre el error, se puede usar $mysqli->error, que devuelve un mensaje de error detallado

Uso y propósito: Este código se utiliza para ejecutar una consulta SQL de tipo **SELECT** en una base de datos MySQL y manejar los resultados. Es útil cuando se desea obtener el número de registros que coinciden con ciertos criterios y manejar posibles errores en caso de que la consulta falle.

Código 26

```php
<?php
```

```php
$conn = new mysqli("localhost", "desarrollador", "adsi2017", "adsi" ) ;
if( $conn->connect_errno) {
echo "Falla al conectarse a Mysql ( ". $conn->connect_errno . ") " .
$conn->connect_error ;
exit() ;
} ;

if($resultado = $conn->query("select codigo, nombre from ciudades ") ){

while($registro = $resultado->fetch_assoc() ){

echo $registro["codigo"] . " " . $registro["nombre"] . "\n" ;

} ;
};

$resultado->free();
$conn->close();
?>
```

Este fragmento de código está escrito en **PHP** y utiliza la extensión **MySQLi** para conectarse a una base de datos MySQL, ejecutar una consulta SQL, recuperar y mostrar los resultados. El código también maneja los errores de conexión y libera los recursos después de completar la consulta. A continuación se explica cada parte del código:

1. Conexión a la base de datos MySQL:

- new mysqli(...): Crea una nueva instancia de la clase mysqli, que se utiliza para interactuar con una base de datos MySQL en PHP.
- Los parámetros proporcionados son:
    - "localhost": El servidor MySQL al que se va a conectar (en este caso, el servidor está en la misma máquina).
    - "desarrollador": El nombre de usuario para la autenticación.
    - "adsi2017": La contraseña asociada al usuario desarrollador.
    - "adsi": El nombre de la base de datos a la que se conecta.

2. Verificación de la conexión:

- $conn->connect_errno: Verifica si hubo un error al intentar conectar a la base de datos. Si ocurre un error, se muestra el código de error y la descripción, y el script termina con exit().
- $conn->connect_error: Proporciona una descripción del error si la conexión falla.

3. Ejecutar la consulta SQL:

- **$conn->query("select codigo, nombre from ciudades")**: Ejecuta una consulta SQL SELECT que recupera los campos codigo y nombre de la tabla ciudades.
- Si la consulta se ejecuta correctamente, el resultado se almacena en la variable $resultado.

4. Recuperar y mostrar los resultados:

- **$resultado->fetch_assoc()**: Obtiene los resultados de la consulta fila por fila como un arreglo asociativo. Cada iteración obtiene una fila de la tabla ciudades con los campos codigo y nombre.

- Dentro del bucle, se imprime el valor de codigo y nombre de cada fila recuperada, seguido de un salto de línea.

5. Liberar los resultados y cerrar la conexión:

- **$resultado->free()**: Libera los recursos asociados con el conjunto de resultados una vez que ya no se necesitan.
- **$conn->close()**: Cierra la conexión a la base de datos, liberando los recursos asociados.

Uso y propósito: Este código se utiliza para ejecutar una consulta SQL que selecciona datos de una base de datos MySQL, mostrar esos datos en la página y gestionar los posibles errores. Es útil para mostrar información desde una base de datos en una aplicación web.

Código 27

```
 if($resultado = $conn->query("select codigo, nombre from ciudades ") ){


    while($registro = $resultado->fetch_object() ){


    echo $registro->codigo . " " . $registro->nombre . "\n" ;
    } ;
};
```

Este fragmento de código en **PHP** utiliza la extensión **MySQLi** para conectarse a una base de datos MySQL, ejecutar una consulta SQL, recuperar los resultados y mostrarlos. En este caso, se utiliza el método fetch_object() para obtener los registros. A continuación, se detalla cada parte del código:

1. Ejecutar la consulta SQL:

- **$conn->query("select codigo, nombre from ciudades")**: Ejecuta una consulta SELECT que obtiene los campos codigo y nombre de la tabla ciudades en la base de datos.
- Si la consulta se ejecuta correctamente, el resultado se almacena en la variable $resultado.

2. Recuperar y mostrar los resultados utilizando fetch_object():

- **$resultado->fetch_object()**: Este método devuelve una fila de resultados como un **objeto**. A diferencia de fetch_assoc(), que devuelve un arreglo asociativo, fetch_object() devuelve un objeto con las propiedades correspondientes a los nombres de las columnas de la tabla (en este caso, codigo y nombre).
- En cada iteración, se imprime el valor de las propiedades del objeto $registro (es decir, codigo y nombre), seguido de un salto de línea.

Uso y propósito: Este código se utiliza para ejecutar una consulta SQL que selecciona datos de la base de datos, procesarlos fila por fila, y mostrar la información. Usando fetch_object(), se accede a los resultados como objetos, lo cual puede ser útil si prefieres trabajar con objetos en lugar de arrays asociativos.

Código 28

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

es utilizada en **PHP** con la extensión **MySQLi** para liberar los recursos asociados con un conjunto de resultados obtenido de una consulta SQL ejecutada previamente. A continuación, te explico el propósito y uso de este método

Propósito y uso de close():

1. Liberar recursos del conjunto de resultados:

- **$resultado->close()**: El método **close()** se utiliza para liberar la memoria y los recursos asociados con el conjunto de resultados obtenido al ejecutar una consulta SQL (por ejemplo, una consulta SELECT).
- Esto es importante para **optimizar el uso de la memoria** y los recursos en el servidor, especialmente cuando se manejan grandes conjuntos de datos o cuando se ejecutan múltiples consultas dentro de un script.
- Aunque **PHP** libera automáticamente los recursos cuando el script termina, es una buena práctica liberar los recursos manualmente después de que ya no se necesiten para mejorar el rendimiento.

2. Cuándo usar close():

- Después de haber terminado de iterar a través de los resultados de una consulta (fetch_assoc(), fetch_object(), etc.), es recomendable llamar a **close()** para liberar el conjunto de resultados.

Código 29

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

es utilizada en **PHP** con la extensión **MySQLi** para cerrar la conexión a una base de datos MySQL. Esta función es importante cuando ya no se necesita interactuar con la base de datos y se quiere liberar los recursos que se están utilizando para esa conexión.

Propósito y uso de close() en el contexto de la conexión:

1. Cerrar la conexión a la base de datos:

- **$mysqli->close()**: Este método se utiliza para cerrar una conexión abierta a una base de datos MySQL utilizando **MySQLi**. Una vez que se llama a este método, la conexión ya no está disponible para realizar consultas, y se liberan los recursos asociados con esa conexión.

2. Es una buena práctica:

- Aunque **PHP** cierra automáticamente las conexiones al finalizar un script, **cerrar explícitamente** la conexión es una buena práctica para **liberar recursos** lo antes posible, especialmente si tu aplicación establece muchas conexiones o si se ejecutan múltiples consultas dentro de un script largo.

3. Uso típico:

- **$mysqli->close()** se debe llamar al final del script o después de que ya no se necesiten más consultas a la base de datos. Esto ayuda a liberar recursos, como el espacio de memoria asociado con la conexión.

Conclusión:

En conclusión, el uso de PHP junto con MySQLi es una combinación poderosa para el desarrollo de aplicaciones web que requieren interacción con bases de datos. Los ejemplos presentados en este informe muestran cómo realizar tareas básicas, como establecer una conexión, ejecutar consultas, manejar los resultados y liberar los recursos de manera adecuada. Además, es importante seguir buenas prácticas, como cerrar explícitamente las conexiones y liberar los recursos utilizados por los conjuntos de resultados para mejorar el rendimiento y la eficiencia de la aplicación. El dominio de estas técnicas es esencial para cualquier desarrollador web que desee crear aplicaciones dinámicas y robustas que manejen grandes volúmenes de datos de manera eficaz y segura.