**PhP**

Este lenguaje nos sirve para imprimir y administrar bases de datos a través de la manipulación de tal en acompañamiento de MySql.

El código PhP se engloba con su inicio y final para ser reconocido:

**<?php** CODIGO **?>**

El comentario en php puede ser de dos formas:

-**//** Este funciona en solo una línea.

-**/\*** comentario **\*/** Este tiene inicio y final, por lo que puede ser de varias filas

En PHP se pueden revalorizar las variables en cualquier momento.

\*El php puede ser escrito dentro del html cuando hablamos de un string

**<h1><?php** código **?></h1>**

\*O el html dentro del php con un echo y sus etiquetas referidas.

**<?php echo “<h1>** código **</h1>” ?>**

Codigos de Texto

**echo “**texto**”; :**Con este código imprimimos texto. También podemos imprimir solo un número.

\*Se puede ingresar elementos de HTML solo en un string. Ejemplo: **“<h1> Hola <h1>”**

\*Los textos se concatenan con un punto . y se les agrega un espacio de forma artificial, con un **“ “.** Puede ser concatenado variables con números en ellas, o números de por si.

Las operaciones tienen que estar concatenadas afuera del **“ ”** y estar entre **( )**

Variables

Las variables se llaman con **$varN.**

\*Su nombre no puede empezar con un número, pero si puede tener luego.

\*Su nombre no puede tener en ningún lugar -.

\*Se imprimen dentro de una string con llaves: {}.

Operaciones

Suma y resta se hace con los botones: **+ -**.

Multiplicación y división se realiza con **\*/.**

Para utilizar un incremento **++** en php primero hay que declararlo y luego llamarlo. Sucede lo mismo con **–**

Se puede aumentar una variable en si misma utilizando el **+=** y revalorizando su contenido.

Utilizar el **%** nos da el resultado restante en una división, por ejemplo en 20/3 este es 2. Nos sirve para detectar pares o inpares, ya que siempre el primero da cero.

\*Para mezclar un string con una operación se utiliza unos paréntesis fuera de ella.

Condicionales

If

Los condicionales son muy parecidos a los de javascript, comenzando con este y luego **{}**. Se puede utilizar también el **else**.

La utilización de **elseif(**nueva condicion**)** nos puede servir para tener condiciones mas completas y detalladas.

\*Si utilizamos una base de datos es recomendabole primero ver si son amyores a 0 y si no hacer un **else** que diga “No resultados”

\*En un array puedo utilizar un **$i + 1 = count($arrayN);** para agregar un mensaje al terminar el conteo.

Switch

Este es como un **if**  pero nos deja utilizar varios casos de forma más fácil sin repetir el **elseif**. Se ingresa la condición que compararemos al lado del **switch**, y luego en cada **case** comparamos el contenido actual con tal caso. Se utiliza un default por si no existe ninguno de los casos.

Se expresa de esta forma:

**switch(**condición que se comparara**) {**

**case** comparación o validación del valor**:**

código**;**

**break;**

**case2** comparación o validación del valor**:**

código2**;**

**break;**

**default:**

codigoDefault;

**break;**

**}**

Array

El array funciona de igual forma que en javascript y se puede escribir de dos formas.

Versiones anteriores y actuales:

**$array1 = [“**contenido1**”, “**contenido2**”, “**contenido3**”, ”**contenido4**”].**

Versiones actuales:

**$array2 = array(“**contenido1**”, “**contenido2**”, “**contenido3**”, ”**contenido4**”)**

**print\_r:** Nos imprime el array completo. Se utiliza entre **<pre> </pre>** en HTML dado que lo ordena.

**var\_dump:** Nos imprime el array completo, pero con mayor información (cantidad de elementos, cantidad de caracteres en cada elemento, si es un numero). Se utiliza entre **<pre> </pre>** en HTML dado que lo ordena.

**$arrayN[] =** nuevo elemento**:** De esta forma agregamos nuevos elementos al arreglo.

**$varN = array.pop($arrayN):** Borra el último elemento del arreglo y lo introduce en una nueva variable.

**$varN = array.shift($arrayN):** Borra el primer elemento del arreglo y lo introduce en una nueva variable.

**$varN = array.splice ($arrayN,** número en el que quieres empezar**,** numero de cuantos te quieres llevar**, array(“**text1**”, “**text2**”)):** Borra partes del arreglo y las reemplaza o agrega con otras.

**unset($arrayN[**N° de elemento**]):** Elimina un elemento del arreglo.

**$varN = in\_array(“**elemento**”, $arrayN):** Revisa si un elemento existe o no en el array. Devuelve 1 si existe, 0 si no.

**array\_key\_exists(“llaveN”, $varN):** Nos dice si una key existe en un array.

**count($arrayN) =** Nos cuenta los elementos de un array. Sirve para **for** o **if.**

**sort($arrayN) =** Ordena los elementos en un orden concreto. Numeros de menor a mayor, string en orden alfabetico.

**rsort($arrayN) =** Ordena los elementos inverso al orden concreto. Numeros de mayor a menor, string en orden contrario alfabetico.

**asort($arrayN) =** Ordena los elementos en un orden concreto en cuanto al valor de las llaves. Numeros de menor a mayor, string en orden alfabetico.

Array Asociativo

Este funciona como los objetos de javascript, teniendo una llave y un valor personalizado. Ejemplo.

**$arrayN = array(**

**“**llave1**”** **=> “**valor1**”,**

**“**llave2**”** **=> “**valor2**”**

**)**

Se ingresan a sus valores a través de sus llaves dentro de corchetes. Ej:

**Echo $arrayN[“**llave1**”]**

**array\_values($arrayN):** Nos regresa los valores en un array con números.

**array\_keys ($arrayN):** Nos regresa las llaves en un array con números.

**$varN = in\_array(“**elemento**”, array\_values($arrayN)):** Revisa si un elemento existe o no en el array. Devuelve 1 si existe, 0 si no.

Array Multidimensional

Este sucede cuando colocamos arrays dentro de arrays asociativos

**$arrayN = array(**

**“**llave1**”** **=> array(**

**“**llaveIn1**”** **=> “**valorIn1”

**),**

**“**llave2**”** **=> array(**

**“**llaveIn2**” => “**valorIn2**”**

**)**

**);**

Se ingresan a sus llaves interiores repitiendo el proceso de nombre dentro de corchetes. Ej:

**Echo $arrayN[“**llave1**”][“**llaveIn1**”]**

ForEach

Recorre los valores de un arreglo para que sea más cómodo imprimirlo.

En un arreglo indentado se utiliza:

**<?php foreach($arrayN as $varN) { ?>**

**<?php** código que se repetirá por cada elemento que haya en el arreglo **?>**

**<?php }; ?>**

O también existe

**<?php foreach($arrayN as $varN): ?>**

**<?php** código que se repetirá por cada elemento que haya en el arreglo **?>**

**<?php endforeach; ?>**

El **$varN** en este caso es una nueva variable que se aplica a todos los elementos y se puede utilizarse para referirse a ellos para imprimirlos o manipularlos.

En un arreglo asociativo se utiliza:

**<?php foreach($arrayN as $key => $val ): ?>**

**<?php** código que se repetirá por cada elemento que haya en el arreglo **?>**

**<?php endforeach; ?>**

El **$key** y el **$val** en este caso son dos variables utilizables que s refieren a la llave y el valor. Nos sirve para imprimirlas o manipularlas.

En un arreglo multidimensional:

**<?php foreach($arrayN[“llaveN”] as $varN): ?>**

**<?php** código que se repetirá por cada elemento que haya en el arreglo **?>**

**<?php endforeach; ?>**

Para ingresar a niveles mas bajos:

**<?php foreach($arrayN as $varN): ?>**

**<?php** **if(array\_key\_exists(“llaveN”, $varN) {?>**

**<?php foreach($varN[“llaveN”] as $varN2): ?>**

**<?php** código **?>**

**<?php endforeach; ?>**

**<?php }?>**

**<?php endforeach; ?>**

Lo que sucede es que debemos primero ingresar al primer nivel, luego para entrar al segundo utilizamos un **if** si existe tal llave. Si existe l recorremos el primer nivel con un **foreach**.

For

Este funciona como en JS y nos sirve para realizar loops y leer o imprimir muchas cosas en repetición. Se escribe:

**for($i = 0; $i < 100; $i++) {**

código que se repetirá.

**}**

Puedo agregarle **if** en medio, pero para que siga la cuenta le tengo que poner **continue**.

Con un **break** rompo el loop.

While

El while es un loop infinito que nos imprime algo hasta que nuestra validación se cumple. Esta validación puede ocurrir fuera del **while**, lo que lo diferencia del **for**. Se escribe:

**<?php $i = 0 ?>**

**<?php while(**validacion**) {**

código que se repetirá**;**

**$i++;**

**}**

**?>**

Tambien puedes utilizar un **if** adentro para imprimir valores si se cumple tal estado.

Function

Son como las funciones de JavaScript, nos permiten manejar diferentes tareas cuando las mandamos a llamar. Pueden tener parámetros. Se escriben:

**Function funN($prm = “**default**”, $prm = “**default**”) {**

Codigo a llamarse

**};**

**funN()**

Se le puede poner un **return** para que nos retonre valores a otra variable o afuera del **function.** Ejemplo:

**Function funN($x, $y) {**

**return x + y;**

**};**

**$resultado = funN(“3”, “4”);**

**echo $resultado;**

Se mostrara 7.

Para agregar valores a un array en una function se tiene que utilizer un **global** dado que este toma valores fuera de la funcion. Ejemplo:

**$agenda = array();**

**function guardarUsuario($nombre, $tel) {**

**global $agenda;**

**$agenda[] = array($nombre, $tel);**

**}**

**guardarUsuario("Antonio", "69");**

**echo "<pre>";**

**var\_dump($agenda);**

**echo "</pre>";**

**Get**

Este código **$\_GET** nos deja tomar valores de los url de otras paginas. Cuando se clickea el url se mandan los datos. Ejemplo:

**<a href="producto.php?id=20&nombre=curso">Ir a Producto</a>**

Otra Página

**<p>Id: </p> <?php echo htmlspecialchars($\_GET["id"]) ?>**

**<p>Nombre: </p> <?php echo htmlspecialchars($\_GET["nombre"]) ?>**

**htmlspecialchars():** Nos sirve para que el código no este “infectado”, y no sea tan vulnerable a hackero.

Formularios

En el inicio del formulario se utiliza **action=”**nombre archivo php**”** y **method=”post”** para mandar los datos en forma de post (se puede con **get**) hacia tal archivo php. Se reciben luego con un **$\_POST[“**nombre de la llave en el arreglo**”]**

Es recomendale asignarle variables a códigos utilizables como **$\_POST**, o a las llaves del arreglo del post.

\*Todos las áreas, select, radio, etc, van adentro del Form.

\*Todos los inputs tienen que tener un **name**. Este puede ser un array, para deominarlo de esta forma solo hay que ponerle un [] luego del nombre elegido.

\*Es recomendable utilizar un **if(isset($\_POST[“submit”]))** teniendo el boton un **name=”submit”** ya que esto verifica que se haya mandado los datos con el botón y no de una forma “ilegal”

Validacion de contenido en Form

Todas estas tienen ciertas cosas en común. Validan si la opción existe primero, como con **isset** o **array\_key\_exist**, esto para no ocupar espacio inútilmente. Luego revisan que contenido tienen y lo imprimen, en el caso de un array con un **foreach**, en el caso de opciones intrínsecascon un **switch**. Cumplen dos fases.

Input:

<?php if(! (filter\_has\_var(INPUT\_POST, "input-name ") &&

                (strlen(filter\_input(INPUT\_POST, " input-name"))> 0))) {

                        //codigo a imprimirse si no hay contenido

                } ?> else {

// código a imprimirse si sí hay contenido

}

Con este código podemos ver si el contenido de un input esta vacio, y reaccionar frente a este.

**Filter\_has\_var:** Este ve si existe una variable en tal input, dado que se crea al momento de colocarle contenido.

**Strlen(filter\_input):** Este esta midiendo que el contenido de la llave en el array sea un espacio con contenido y no espacio en blanco, el filet\_input es para poder acceder al contenido y no a la llave.

Checkbox singular:

<?php if(isset($\_POST["checkbox-name "])) {

                        $checkbox-var = $\_POST["checkbox-name];

                        if($checkbox-var =="on") {

                // código a imprimirse si esta activo el checkbox

    } else {

// código a imprimirse si no esta activo el checkbox

}

} ?>

Nos fijamos si existe tal parte en el array, y si asi es, vemos si esta prendida o no con un **if**. En tal caso hacemos un código especial.

Checkbox array:

Se debe poner en el name del conjunto de checkboxes su nombre mas un [] asi todos están en el mismo array y están relacionados. Por ejemplo: **name=”array[]”** y asi en todos.

Para diferenciarlos utilizamos la propiedad **value** y ahí si les colocamos nombres diferentes a cada uno.

 <?php if(isset($\_POST["checkboxes-name"])) {

                        $checkboxes-var = $\_POST["checkboxes-name"];

echo “checkboxes en uso”

                 foreach($checkboxes-var as $new-variable){

                                echo $new-variable . "</br>";

                        }

                }

        ?>

Select:

Primero debemos darle al select un **name**, y luego también a cada **option** un **value** diferente. Como el select no crea un array, solo hay que ver cada opción que tiene y crear un caso para cada una de ellas.

 <?php  if(isset($\_POST["select-name"])) {

                        $select-var = $\_POST["select-name"];

                        echo "<h2> Selects </h2>";

                        switch($select-var) {

                                case "option-value":

//codigo a imprimirse

                                break;

                                case "option-value":

                                         //codigo a imprimirse

                                break;

                                case "option-value ":

                                        //codigo a imprimirse

                                break;

                                default:

                                        //codigo a imprimirse

                                break;

                        }

                } ?>

Radio:

Este es mas complicado ya que su propio html necesita de la presencia de un php. El cual es:

<div class="tipo\_curso">

                  <h2>Tipo de Curso</h2>

                  <?php  $array-name = array(

                      'key1' => ‘opcion1’,

                      'key2' => ‘opcion2’

                  );?>

                  <?php foreach($array-name as $key => $opcion) {

      echo "<div class='campo'>";

      echo "<input type='radio' name='opciones' value='$key' > $opcion";

      echo "</div>";

                  } ?>

            </div>

En este creamos un array con las opciones dentro de llaves que luego utilizaremos para facilitar la verificación. Luego creamos un **foreach** que recorra tal arreglo e imprima las opciones en un input. El value son las **$keys** mientras que cada opción se muestra.

Luego debemos checar si existe la key del opciones (si no, entonces no se eligió ningún radio). Y como funciona como un select (donde solo hay siempre una opción marcada concreta) se puede recorrer con un switch.

<?php if(array\_key\_exists($\_POST["array-name"], $array-name)) {

                          $array-var = $\_POST["array-name"];

                          switch($array-var) {

                                  case "key1":

                                        //codigo a imprimirse

                                  break;

                                  case "key2":

                        //codigo a imprimirse

break;

                          }

                  } else {

                                //codigo a imprimirse

                          } ?>

TextArea:

 <?php if(isset($\_POST["textarea-name"])) {

                        $textarea-var = $\_POST["textarea-name"];

                  if(strlen($textarea-var) > 0 && trim($textarea-var)) {

                                echo "<h2> Mensaje </h2>";

                                echo $textarea-var;

                        } else {

                                echo "No existe Mensaje";

                        }

                } ?>

Como es casi igual a un input se valida dew forma casi igual, primero con un **isset** y luego si el mensaje mide mas que 0 y borra los espacios en blanco (**trim**). Luego el contenido solo se llama con la propia función del **$\_POST;**

**Seguridad en forms**

Ya que el usuario puede mandar dentro de inputs o textareas scripts con códigos que infecten nuestra pagina, debemos “curar” el contenido de tales.

Hay muchas formas de hacer esto, dos de las que veremos son a través de tomando la var del contenido del **$\_POST**, dándole un nuevo nombre, y aplicándole funciones propias de php.

Ejemplo 1:

 <?php if(isset($\_POST["textarea-name"])) {

                        $textarea-var = $\_POST["textarea-name"];

$textarea-seguridad = filter\_var($textarea-var, FILTER\_SANITIZE\_STRING)

 if(strlen($textarea-seguridad) > 0 && trim($textarea-seguridad)) {

                                echo "<h2> Mensaje </h2>";

                                echo $textarea-seguridad;

                        } else {

                                echo "No existe Mensaje";

                        }

                } ?>

Ejemplo 2:

<?php if(isset($\_POST["textarea-name"])) {

                        $textarea-var = $\_POST["textarea-name"];

$textarea-seguridad = htmlespecialchars($textarea-var)

 if(strlen($textarea-seguridad) > 0 && trim($textarea-seguridad)) {

                                echo "<h2> Mensaje </h2>";

                                echo $textarea-seguridad;

                        } else {

                                echo "No existe Mensaje";

                        }

                } ?>

Cortar Partes de paginas y

organizarlas a través de php

Este mecanismo es para orgnizar de mejor forma nuestros archivos, creando unos folders en donde tendremos distintas partes de la pagina. Estos se ubicarían en un ejemplo en la siguiente categorización.

**-Includes**

--Templates

---header.php

---main-section.php

---footer.php

--Functions

Dentro del template están las partes de código html y php, y en function son las relaciones entre php y la base de datos, o funciones más complejas de back-end.

Estos templates nos sirven para reutilizarlos en otras páginas que no sean el index de por sí y no tener que ir cambiando nuestro código en cada una de ellas si necesitamos hacer un cambio. Ahorramos código idéntico.

Incluimos partes separadas de archivos con el código:

**<?php include\_once ‘**path/to/section.php**’ ?>**

\*Este debe estar dentro del archivo php donde la queremos colocar, el archivo section tamb debe ser php

\*Es importante posicionar este codigo en la parte de nuestro index donde debería ir si es colocado en función cascada.

**Manipulacion de la base de datos de Mysql**

Conectar a la base de datos

En la carpeta functions creada anteriormente creamos un nuevo archivo php en el cual haremos nuestra conexión a la base de datos.

Conexión Simple:

**<?php**

**$conn = new mysqli (“localhost”, “**usuario-wamp**”, “**contraseña-phpmyadmin**”, “dtbN”)**

**?>**

Sabemos el estatus de la conexión con:

**<?php**

**If($conn->connect\_error) {**

**echo $error -> $conn->connect\_error**

**}**

**?>**

O

**<?php**

**echo $conn->ping();**

**?>**

Conexión Específica:

<?php

//Credenciales de las Bases de Datos

define("DB\_USUARIO", "root");

define("DB\_PASSWORD", "root");

define("DB\_HOST", "localhost");

define("DB\_HOST", "agendaphp");

$conn = new mysqli(DB\_HOST, DB\_USUARIO, DB\_PASSWORD, DB\_NOMBRE);

?>

Comunicarnos con la base de datos

Esto lo realizamos en un archivo diferente al **db\_connection** con un **try-catch**, el cual se comunica constantemente con la base de datos y si falla la página no crasheara op se verá afectada:

**<?php**

**try {**

**require\_once(“**path/to/db\_connection.php**”)**

**$sql = “** CODIGO MYSQL **”**

**$consulta = $conn->query($sql)**

**} catch (\Throwable $th) {**

**$th->getMessage();**

**}**

**?>**

**<?php**

**$conn->close();**

**?>**

Dentro del try escribimos nuestro código y en el catch lo que pasaría si no funiona. El $th es una variable realizada para crear un mensaje de error.

**require\_once():** Al igual que el **include\_once** este nos comunica con otro archivo, en este caso el php de conexión al server, asi podemos comunicarnos con tal.

**$sql =**Esto nos sirve para que la consulta sea más fácil de comprimir y mover**.** No es necesario colocarle una variable al código mysql ya que php lo soporta.

\*Esta var puede cambiar de nombre.

**$consulta = $conn->query($sql)** En esta parte del código realizamos la consulta a través de **query** comunicándonos con la db utilizando la variable **$conn.** Dentro de la consulta va nuestro código a realizarse.

\*Esta var puede cambiar de nombre.

**$conn->close();** Cerramos la conexión. Esto evita que se sobrecarge mysql (cerrarlo luego de la impresion).

\*En un resumen lo que hacemos es realizar la conexión, escribir la consulta, mandar la consulta a la bd y luego imprimir el resultado.

Impresión de arreglos (o tablas) de la bd

en nuestra pagina

Para imprimir cualquier contenido de nuestra base de datos primero debemos hacer todos los pasos anteriores mencionados. Luego a travez de las var escritas realizamos, en un ejemplo de impresión de todos los datos, en un array:

**<div>**

**<?php**

**$varN = $consulta->fetch\_assoc();**

**var\_dump($varN)**

**?>**

**</div>**

De esta forma la consulta que estábamos realizando le damos una nueva variable que será impresa en nuestra página.

**Fetch\_assoc:** Imprime los resultados en forma de array, por ahora solo el primero.

**fetch\_array(MYSQLI\_ASSOC):** Otra forma de traer los resultados en un array.

\*Para imprimir todos los resultados debemos colocarlo en un **while**

**<div>**

**<?php**

**while( $varN = $consulta->fetch\_assoc() ) {**

**};**

**var\_dump($varN)**

**?>**

**</div>**

**Impresión de arreglos especificos**

Dado que el **fetch\_assoc** nos coloca los datos en un array debemos consultar sus datos como antes recorríamos un array. En este caso:

**$varN[“**llaveN-columnaN**”]**

**Impresión de arreglos en tablas relacionadas.**

Para esto debemos modificar nuestra consulta cuando la realizamos en la var **$sql.** Debe estar seleccionando cada columna en especifico y en el caso de la relación, NO utilizar la columna de relación principal, sino las que estas dentro de tal. Por ejemplo imaginemos que tenemos un db tal:

Tabla **General**

Columna **relacion\_id=** datos **1/2/**

Tabla **Secundaria**

Columna **id\_secundaria = 1/2/**

Columna **nombre=** datos **juan/Antonio/lucas**

Columna **apellido=** datos **glorioso/Galarza/bottino**

Entonces dado que solo nos comunicamos con la tabla general no seleccionamos su columna de relación, sino las columnas dentro de la tabla relacionada:

**$sql = “ SELECT nombre, apellido “;**

Luego debemos concatenar el código con las consultas que ven de que tablas sacamos los datos, en **$sql:**

**$sql .= “ FROM general ” ;**

**$sql .= “ INNER JOIN secundaria ” ;**

**$sql .= “ ON general.relacion\_id = secundaria.id\_secundaria “**

Lo que hacemos es primero elegir la tabla general, relacionarla con la secundaria, y luego unificar las columnas que están comunicando las relaciones a través de las id.

Formatear los arrays (o tablas)

Esto se hace para poder organizar nuestros arrays de datos en distintas maneras que queramos, dependiendo la información que tengan en ellos.

Para esto debemos crear un array vacio fuera del **while** (el que creamos para imprimir con el **fetch**) para poder meter en este toda la información.

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**};**

**var\_dump($varN)**

**?>**

**</div>**

Luego creamos un arreglo asociativo dentro del **while** que va a contener los datos que querramos organizar

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**};**

**var\_dump($arrayN2)**

**?>**

**</div>**

De esta forma filtramos la información, pasando los datos de un arreglo menos arreglado a otro con nuestra seleccion a gusto.

Luego podemos colocar este arreglo en el anterior creado, así unimos todos los datos en una organización mayor:

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**$arrayN[] = $arrayN2;**

**};**

**var\_dump($arrayN)**

**?>**

**</div>**

Filtrar los datos en nuevos arreglos

Dado a la anterior forma podemos crear nuevas formas de filtrar el contenido teniendo solo los datos que queremos.

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$varN2 = $varN[“llaveDeDato”] ;**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**$arrayN[$varN2][] = $arrayN2;**

**}; //termina el while**

**var\_dump($arrayN)**

**?>**

**</div>**

Y así creando una nueva variable que solo contiene una llave con un dato, pasándola al array más grande y foramteando todo en el arrayN2, podemos filtrar cualquier dato.

Impresión de los datos de la db

Este paso es para dejar de lado la impresión en formato arreglo y pasar nuestros datos ya ahora de una forma más limpia a la página.

Lo primero que debemos hacer es recorrer el arreglo multidimensional con un **foreach**. \*El primer **$varN** es a la vez un **$arrayN**, no porque lo implique el código sino dado que los datos vienen en esa forma de la database, acomodados por el fetch\_assoc.

\*Solo colocar en el while las partes html necesarias para la impresión de los datos, todo los divs o section que los rodean AFUERA DE TAL.

\*Si solo tenemos un arreglo unidimensional y queremos llegar solo a los datos, con un **$varN[“llave-N”]** basta.

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$varN2 = $varN[“llaveDeDato”] ;**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**$arrayN[$varN2][] = $arrayN2;**

**}; //termina el while**

**foreach($arrayN as $varN3 => $varN4) {**

**echo $varN3[“llaveN”];**

**echo $varN4[“datoN”];**

**}//Recorrido 1era Dimension**

**?>**

**<?php var\_dump($arrayN) ?>**

**</div>**

Esta sintaxis nos sirve para acceder a las llaves y a los datos de igual forma. Siendo **$varN3** la primera llave y **$varN4** los arrays dentro de esta. Recorremos de esta forma solo una dimensión del arreglo.

Para recorrer la segunda dimensión solo necesitamos crear otro **foreach**.

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$varN2 = $varN[“llaveDeDato”] ;**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**$arrayN[$varN2][] = $arrayN2;**

**}; //termina el while**

**foreach($arrayN as $varN3 => $varN4) {**

**echo $varN3[“llaveN”];**

**echo $varN4[“datoN”];**

**foreach($varN4 as $varN5 => $varN6) {**

**echo $varN5[“llaveN”];**

**echo $varN6[“datoN”];**

**}//Recorrido 2da Dimension**

**}//Recorrido 1era Dimension**

**?> <?php var\_dump($arrayN) ?>**

**</div>**

De esta forma podemos imprimir las llaves y datos del segundo arreglo, llegando a la 2da dimensión del principal arreglo.

Multi-Consultas

Si realizamos muchas consultas, ya sea a diferentes tablas, o dado a que les colocamos diferentes restricciones cuando las pedimos a cada una, debemos modificar nuestra forma de llamarlas e imprimirlas:

Llamado a la base:

**<?php**

**try {**

**require\_once(“**path/to/db\_connection.php**”)**

**$sql = “**consulta1**;”**

**$sql .= “**consulta2**;”**

**$sql .= “**consulta3**;”**

**} catch (\Throwable $th) {**

**th->getMessage();**

**}**

**?>**

**<?php $conn->multi\_query($sql) ?>**

Impresión de los datos:

**<?php**

**do {**

**$resultado = $conn->store\_result();**

**$row = $resultado->fetch\_all(MYSQLI\_ASSOC);**

**//Codigo html y php**

**<?php $resultado->free();?>**

**} while ($conn->more\_results() && $conn->next\_result() ); ?>**

**<?php**

**$conn->close();**

**?>**

**\***Tener en cuenta que la consulta viene en un array, el código html y php podría necesitar estar rodeado entre un **foreach**.

Resumen Base De Datos Manipulación

En resumen lo que haces es realizar tu consulta con un **query**, luego la respuesta la manipulas formateándola en otros **arrays** (si es de tu gusto) y por ultimo imprimes los datos de los nuevos **arrays** con los **foreach.** Una consulta semi-compleja singular entonces quedaría:

**<?php**

**try {**

**require\_once(“**path/to/db\_connection.php**”)**

**$sql = “** CODIGO MYSQL **”**

**$consulta = $conn->query($sql)**

**} catch (\Throwable $th) {**

**$th->getMessage();**

**}**

**?>**

**<div>**

**<?php**

**$arrayN = array();**

**while( $varN = $consulta->fetch\_assoc() ) {**

**$varN2 = $varN[“llaveDeDato”] ;**

**$arrayN2 = array(**

**“llaveN” => $varN[“llaveDeDato”] ,**

**“llaveN2” => $varN[“llaveDeDato2”]**

**)**

**$arrayN[$varN2][] = $arrayN2;**

**}; //termina el while**

**foreach($arrayN as $varN3 => $varN4) {**

**echo $varN3[“llaveN”];**

**echo $varN4[“datoN”];**

**foreach($varN4 as $varN5 => $varN6) {**

**echo $varN5[“llaveN”];**

**echo $varN6[“datoN”];**

**}//Recorrido 2da Dimension**

**}//Recorrido 1era Dimension**

**?> <?php var\_dump($arrayN) ?>**

**</div>**

**<?php**

**$conn->close();**

**?>**

Inserción de datos a la database.

Statement Prepared

En este caso usaremos el código **mysqli.** Para hacerlo debemos hacerlo en la pagina donde recibamos los datos, ya sea atravez de un **$\_POST** u otro método. Luego creamos un **trycatch** con la conexión a la **db.**

**<?php**

**$dato1 = $\_POST[“llaveN”]**

**$dato2 = $\_POST[“llaveN”]**

**try {**

**require\_once(“**path/to/db\_connection.php**”)**

**$stmt = $conn->prepare(“INSERT INTO tablaN (columnaN, columnaN2) VALUES (?,?)”);**

**$stmt->bind\_param(“si”, $dato1, $dato2);**

**$stmt->execute();**

**$stmt->close();**

**$conn ->close();**

**header("Location: thisarchive.php?exitoso=1");**

**} catch (\Throwable $th) {**

**$th->getMessage();**

**}**

**?>**

**\***Si estamos pasando un valor en string, en el **bind\_param** va una **s**, en cambio sí es un entero va una **i**. Esto dado a que nuestras columnas recibirán datos que sean iguales a solo los que pueden recibir, como **INT** o **VARCHAR.**

\*Dado a que pasar datos en un array en una base de datos es un proceso más complicado, es más simple convertir tales a archivos **json**. Esto se lograría poniendo en la columna de la tabla que espere un **LONGTEXT** y que todos los datos o llaves con datos, vengan como objetos en texto.

\*El **header()** nos evita que se manden muchas consultas iguales a nuestra **database.** Para que este funcione TODO el código php que utilizamos para la incersion debe estar antes de que siquiera se cargue el html (es decir antes de la primera etiqueta **<html>**)

Crear archivos Json

Estos sirven para decodificar **arrays** que recibamos de un **form** y poder colocarlos en la base de datos de forma menos complicada, ya que estos transforman **arrays** en objetos. Para hacerlo primero debemos crear una función que reúna las variables con los datos en argumentos. Ejemplo:

**Function ejemplo\_json (&$varN, &$varN2, &$varN3,) {**

**$json = array()**

**Foreach ($varN as $key => $data) {**

**$json[key] = $data;**

**};**

**$varN2 = (int) $varN2;**

**if ($varN2 > 0) {**

**$json["varN2"] = $varN2;**

**}**

**foreach($eventos as $evento) {**

**$eventos\_json["evento"][] = $evento;**

**}**

**return json\_encode($json);**

**}**

\*Siendo **varN** un **array** y **varN2** una llave con un dato.

Lo que estamos haciendo primero es crear un **array json** que se le hara **encode**.

Si la variable tiene muchas llaves pero con datos únicos se deben recorrer y colocarlos en el **json**, cada uno como su propio objeto (**varN**).

Luego si la variable es una sola llave y solo contiene un dato, este lo ponemos en el **array** del **json** creando su propia llave y colocándole el valor único (**varN2**).

Luego si la variable es una sola llave pero con muchos datos, se crea una llave en el **json** y ahí se colocan sus datos utilizando un **[]** (**varN3**).

Por último se hace un **return** del array como el archivo json.

Esta función se debe llamar en el archivo donde les damos los valores del array a las variables y además se debe crear un **include** que lo conecte a donde creamos las funciones. Ejemplo:

**$varN = $\_POST[“llaveN”];**

**$varN2 = $\_POST[“llaveN2”];**

**$varN3 = $\_POST[“llaveN3”];**

**Include\_once “path/to/functions.php”**

**$varN4 = ejemplo\_json($varN, $varN2, $varN3)**

**Echo $varN4**

\*El **&** se utiliza para lo que se llama pasar por referencia, una accionq ue ayuda a mantener los datos intactos.

Funciones Random

**date(“**como desea ser representado/ver php guia**”, strtotime($varN)):** Formateamos una fecha para imprimirla, dado que solo reconoce el formato fecha debemos cambiar el string que tengamos en el **$varN** a time (**strtotime**).

**setlocale(LC\_TIME, “spanish”):** Cambia como el servidor interpreta string y dates, pasándolas a español.

**$arrayN3 = array\_combine($arrayN, $arrayN2):** Con este metodo combinamos dos arrays, pudiendo colocar los valores de la primera como las llaves de la segunda.

**(int) $varN:** Transforma un string a números.

**strftime (“**como desea ser representado/ver php guia**”, strtotime($varN)):** Formateamos una fecha para imprimirla, dado que solo reconoce el formato fecha debemos cambiar el string que tengamos en el **$varN** a time (**strtotime**). A diferencia del **date** si recibe cambios a través del **setlocale.**

**strtolower():** Devuelve el string en todo minúsculas.

\*Si necesito imprimir algo cada número par usar:

<?php if($i % 2 == 0) { ?>

 //Code

<?php } ?>

\*Si necesito imprimir algo cada número impar usar:

<?php if($i % 2 == 1) { ?>

 //Code

<?php } ?>