POLITÉCNICO DE COLOMBIA
educación sin límites

Diplomado en:

Programación en PHP

Guía didáctica Nº 5



Formación Virtual

educación sin límites

GUÍA DIDÁCTICA N°5
M2-DV59-GU05
MÓDULO 5: DESARROLLO WEB II
DIPLOMADO EN PROGRAMACIÓN EN PHP

© DERECHOS RESERVADOS - POLITÉCNICO DE COLOMBIA, 2019
Medellín, Colombia

Proceso: Gestión Académica
Realización del texto: Diego Palacio, Docente
Revisión del texto: - Jehison Posada, Asesor Gramatical
Diseño: Cristian Quintero, Diseñador Gráfico

Editado por el Politécnico de Colombia

ÍNDICE

PRESENTACIÓN 4
COMPETENCIAS 5
TEMA 1 Bases de Datos - MySQL 6
TEMA 2 PHP - PDO 32
TEMA 3 DESARROLLO DEL PROYECTO - ESTRUCTURA 58
TEMA 4 DESARROLLO DEL PROYECTO - FORMULARIOS 63
TEMA 5 DESARROLLO DEL PROYECTO – MVC 63
TEMA 6 DESARROLLO DEL PROYECTO – FUNCIONALIDADES - 1 63
ASPECTOS CLAVES 66

PRESENTACIÓN

La Guía Didáctica N°5 del MÓDULO 5: DESARROLLO WEB II, es un material que ha sido desarrollado para el apoyo y orientación del participante en el *Diplomado en Programación en PHP*, especialmente, está orientada a la consolidación y/o desarrollo de las habilidades y destrezas en el desarrollo web bajo este lenguaje.

Como bien conoces, el objetivo principal de este módulo número 5 es profundizar un poco en el concepto de la programación web bajo PHP para crear unas sólidas bases en la implementación e integración de este lenguaje con otras tecnologías.

Para ello, se ha organizado esta guía ocho (8) contenidos temáticos, entre ellos: (a) Bases de Datos - MySQL, (b) PHP - PDO, (c) Desarrollo del Proyecto – Estructura, (d) Desarrollo del Proyecto - Formularios, (e) Desarrollo del Proyecto – Clases, (f) Desarrollo del Proyecto – MVC, (g) Desarrollo del Proyecto – Funcionalidades, y (h) Resumen Final del Curso, que ayudarán a formar unas buenas cualidades y prácticas de la programación web en PHP.

COMPETENCIAS

Se espera que con los temas abordados en la Guía Didáctica N°5 del MÓDULO 5: DESARROLLO WEB II, el estudiante logre la siguiente competencia:



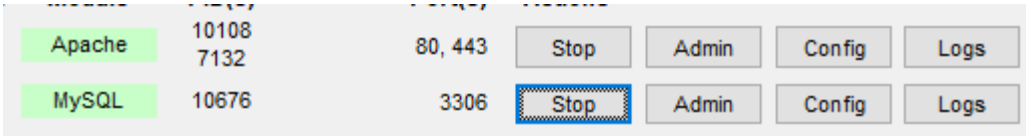
- Desarrollar de forma completa un sistema de información basado en el lenguaje de programación PHP y MySQL como motor de bases de datos.

Indicadores de logro:

- Crear una base de datos en phpMyAdmin.
- Administrar una base de datos en MySQL
- Crear y administrar tablas en MySQL.
- Insertar, consultar, actualizar y eliminar registros de una base de datos.
- Realizar copias de seguridad.
- Generar diagramas de modelo entidad relación.
- Crear una conexión a una base de datos en MySQL por medio de PHP.
- Entiende el patrón de diseño Modelo Vista Controlador.
-

TEMA 1
Bases de Datos - MySQL

Al momento de descargar XAMPP, este ofrecía dos características fundamentales en el desarrollo bajo PHP. El primero fue Apache, el servidor que se ha utilizado a lo largo del diplomado para ejecutar nuestro código PHP en el navegador, y ahora el otro, MySQL o en otras palabras, phpMyAdmin:



phpMyAdmin es una herramienta gratuita, que permite de una manera muy completa acceder a todas las funciones de la base de datos MySQL a través de PHP, mediante una interfaz web muy intuitiva.

¿Qué ofrece phpMyAdmin?

Esta herramienta es muy completa y nos ofrece una gran cantidad de usos y características, algunas de ellas son:

- Esta aplicación nos permitirá realizar las operaciones básicas en base de datos MySQL, como son: crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar sentencias SQL, administrar claves de campos, administrar privilegios y exportar datos en varios formatos. La función de exportar datos se emplea muchas veces para realizar backups de la base de datos y poder restaurar esta copia de seguridad en el futuro a través de phpMyAdmin mediante la opción "importar".

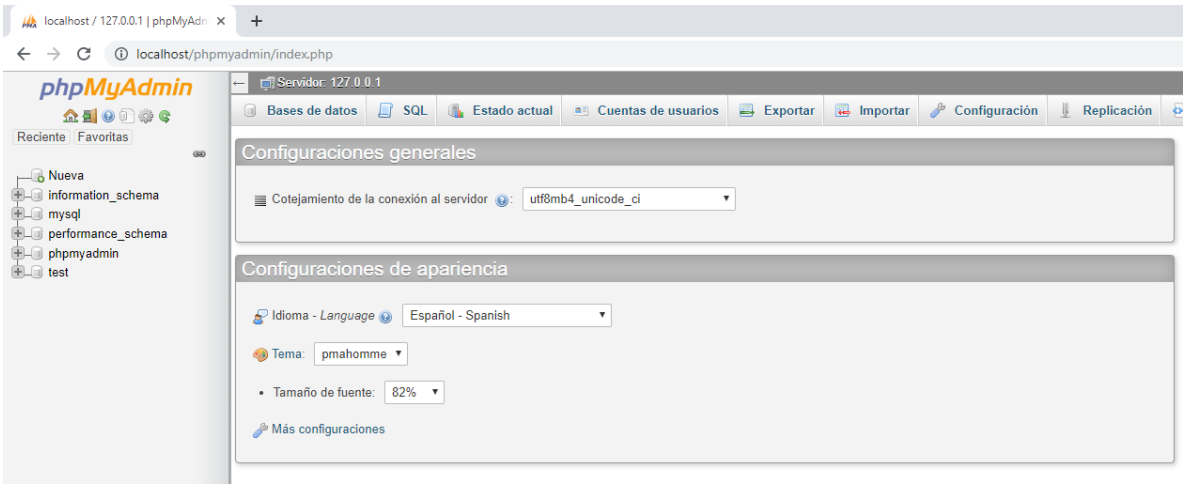
- phpMyAdmin es el administrador de bases de datos por defecto en muchos paneles de control web comerciales como son cPanel, Plesk o DirectAdmin.

- Los usuarios no deberían tener problemas a la hora de manejar esta herramienta, ya que es fácil de usar.

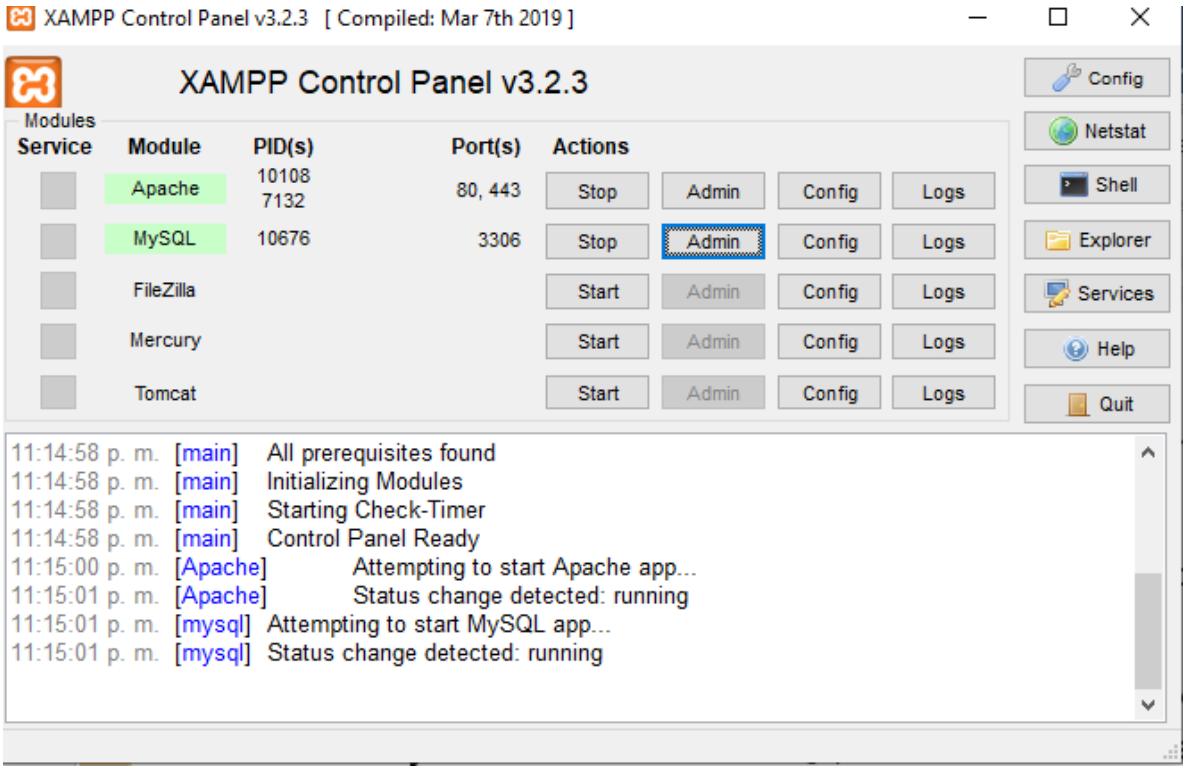
- Otra de las funciones más importantes que nos ofrece es que permite optimizar y reparar tablas, las cuales son dos tareas de mantenimiento fundamentales.

- Nos da la posibilidad de realizar búsquedas en las bases de datos, además de poder escribir nuestras propias consultas SQL de manera directa y ejecutarlas.

- Esta herramienta también es de gran ayuda para desarrolladores de aplicaciones que empleen MySQL, ya que permite depurar consultas y hacer tests de forma rápida y sencilla.



Las formas de acceder a phpMyAdmin son muy sencillas, la primera de estas es directamente por medio de XAMPP, dando click en la opción admin:



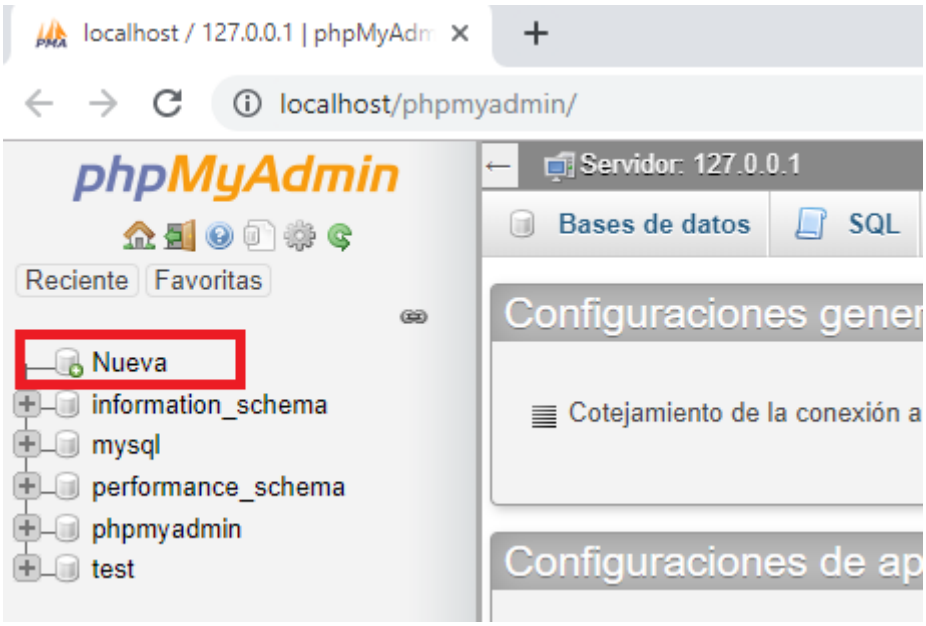
O yendo directamente a la ruta donde se ejecuta phpMyAdmin en: http://localhost/phpmyadmin/index.php.

Esta herramienta provista por XAMPP tiene una gran cantidad de características y funciones esenciales descritas anteriormente, pero el alcance del diplomado estará relacionado con los siguientes temas:

- Creación de bases de datos.
- Creación de tablas.
- Especificación de campos de tabla.
- Tipos de datos.
- Claves primarias.
- Inserción de datos.
- Consulta de datos.
- Actualización de datos.
- Eliminación de datos.
- Exportación de datos y tablas.
- Copias de seguridad.
- Modelo Entidad Relación.

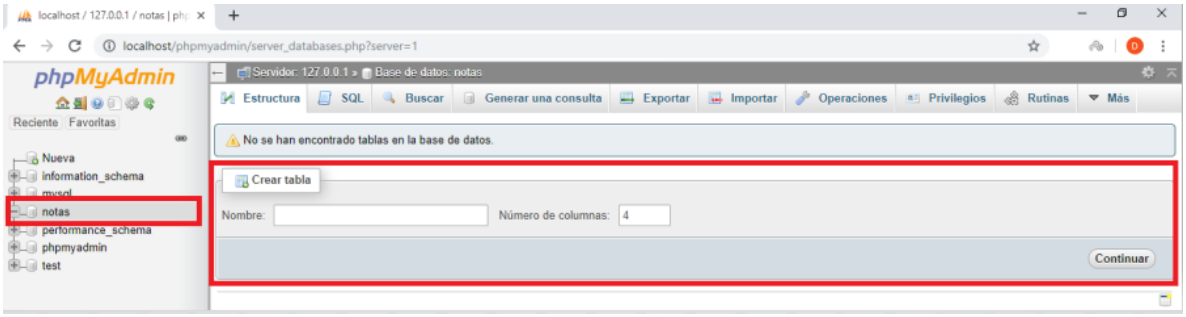
Para crear nuestra primera base de datos, que será la utilizada en el proyecto final, el proceso que debemos realizar es el siguiente.

- Abrir phpMyAdmin.
- En el apartado superior izquierdo, se encuentran las bases de datos establecidas por defecto de MySQL y las que el desarrollador vaya ocupando en su labor. Ir a la opción de "Nueva".



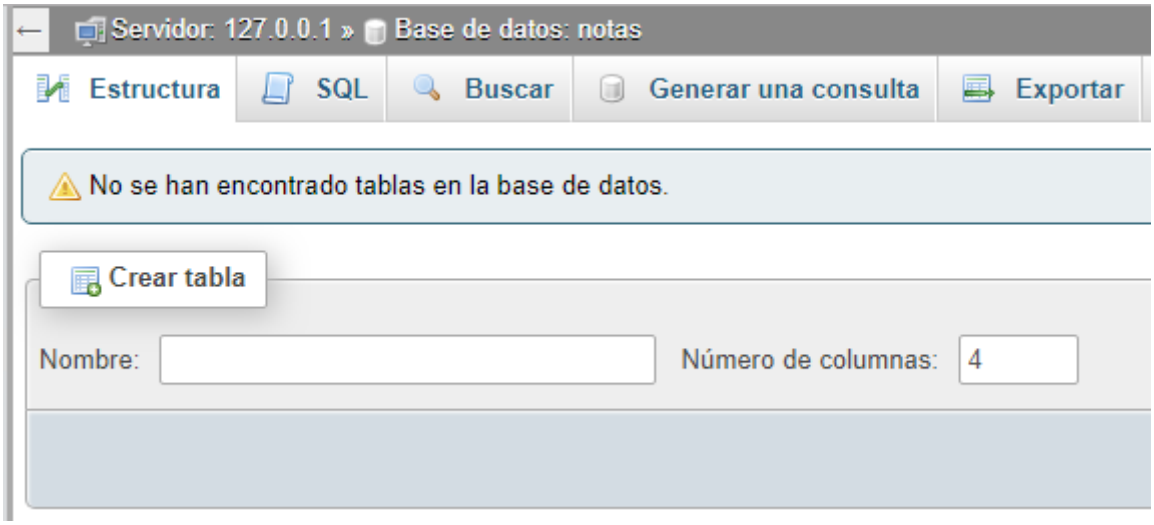
- En esta nueva ventana, tendremos dos opciones para continuar, el nombre de la base de datos y el cotejamiento (Codificación de datos). Por defecto serán los siguientes serán:



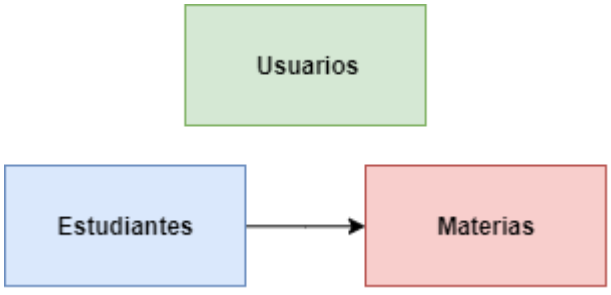


A partir de este momento empezaremos el diseño de nuestro proyecto a partir de los temas previstos en los temas de este módulo, iniciando por nuestra base de datos.

Contamos en este momento con una base de datos de nombre "Notas":



En base a nuestra base de datos y más específicamente nuestro proyecto, contaremos con 4 tablas en la siguiente estructura:



Nota: *Para el desarrollo de este proyecto tendremos algunas "malas prácticas en el desarrollo del proyecto, por lo complejo de explicar conceptos como inner joins, claves foráneas y demás. Pronto habrá un diplomado centrado únicamente en el desarrollo de bases de datos en MySQL.*

Tabla de Usuarios

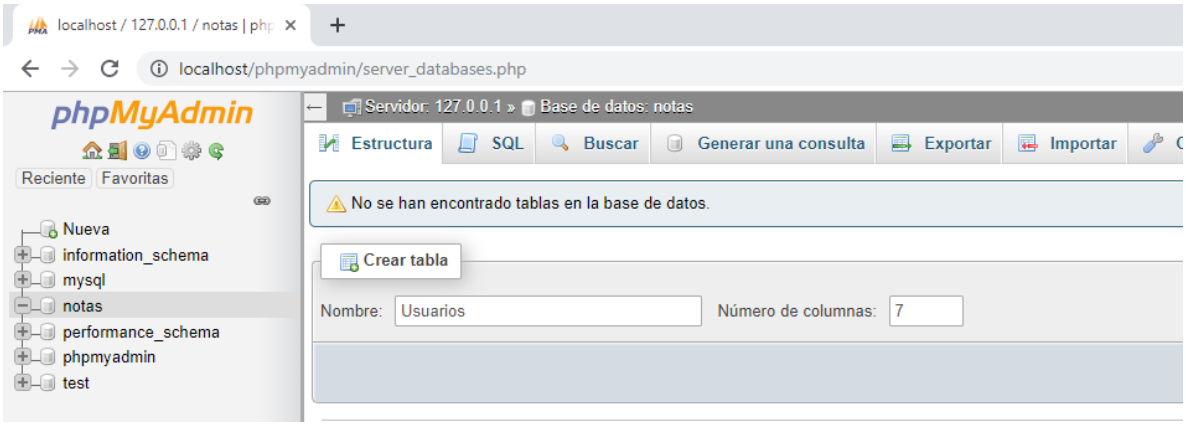
La tabla usuarios dentro de nuestro sistema, contendrá la información de los respectivos usuarios que podrán ingresar a nuestro sistema. La información que contendrá nuestra tabla será:

Usuarios						
ID_USUARIO	**NOMBRE**	**APELLIDO**	**USUARIO**	**PASSWORD**	**PERFIL**	**ESTADO**
1	Diego	Palacio	dpala1123	123	Administrador	Activo
2	Stiven	Roggers	strr42	321	Docente	Activo

La tabla contará con la siguientes características e información para su correcta implementación.

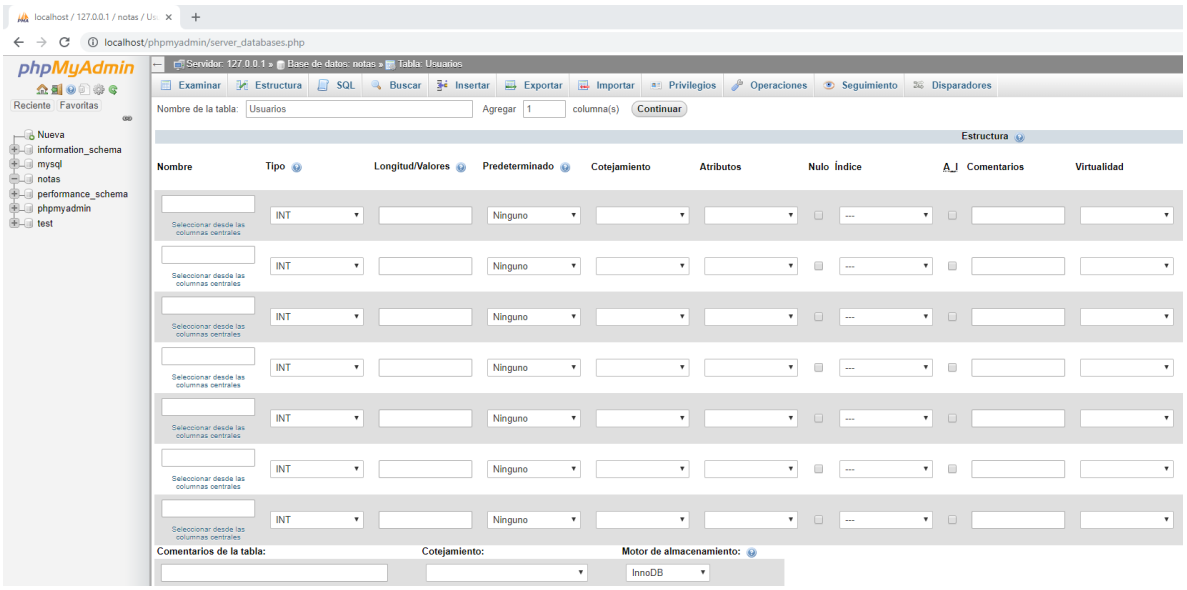
- El nombre de la misma será Usuarios, el nombre es claro y diciente de la información que contendrá.
- Tiene 7 campos, con nombres claros y en singular.
- Cada campo deberá contar con una tipificación. Algo similar a PHP con los tipos de datos.
- Cada campo deberá contener un tamaño máximo de almacenamiento (caracteres).
- El primer campo por estándar para el diplomado será el ID y a su vez clave primaria (Un campo único en la tabla que permitirá reconocer y diferencia todos los registros. Ej: El Id 1, pertenece a toda la información de Diego y nunca habrá otro Id 1).
- Otra característica del ID es que será auto_increment, es decir, cada vez que se registre un nuevo registro (valga la redundancia) el campo ID aumentará en razón de 1.

Para realizar la creación de la tabla "Usuarios" dentro de la base de datos "notas", regresemos al apartado de phpMyAdmin donde quedamos:



- Debemos se reconocer que estamos en la base de datos donde se realizará la creación de la tabla.
- Debemos especificar el nombre de la tabla: "Usuarios".
- Debemos especificar el número de columnas (7: Id, Nombre, Apellidos, Usuario, Password, Perfil y Estado).

Obtendremos este resultado, una interfaz para detallar cada uno de los campos de nuestra tabla.

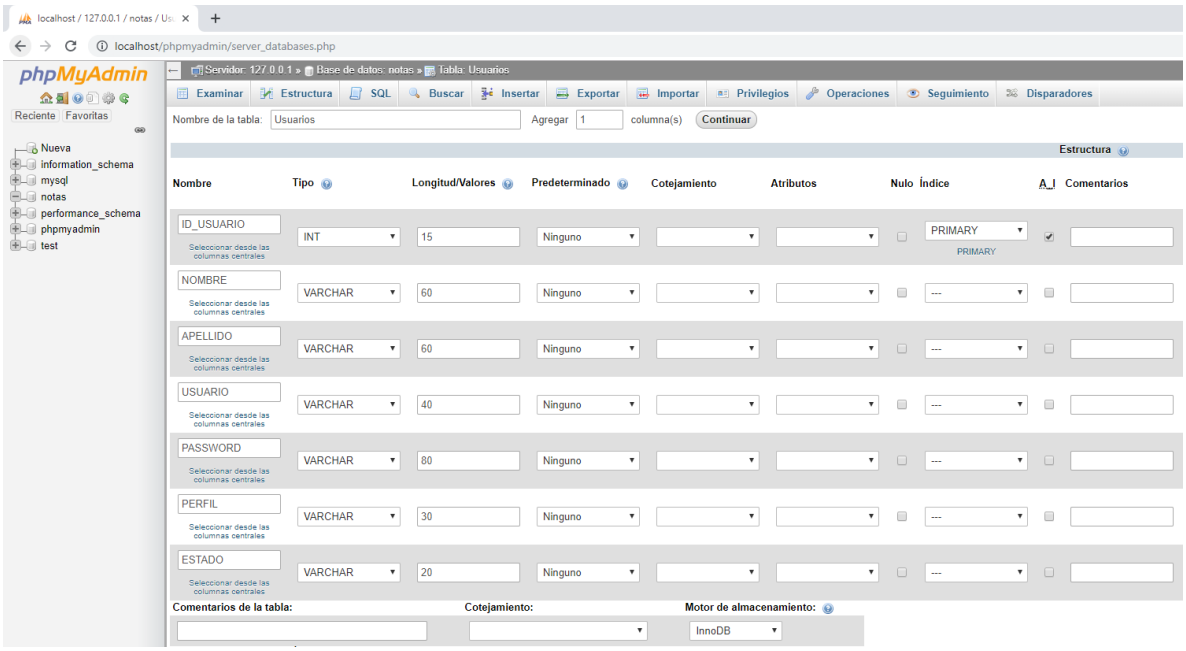


Definiendo las características de nuestra tabla, serían las siguientes:

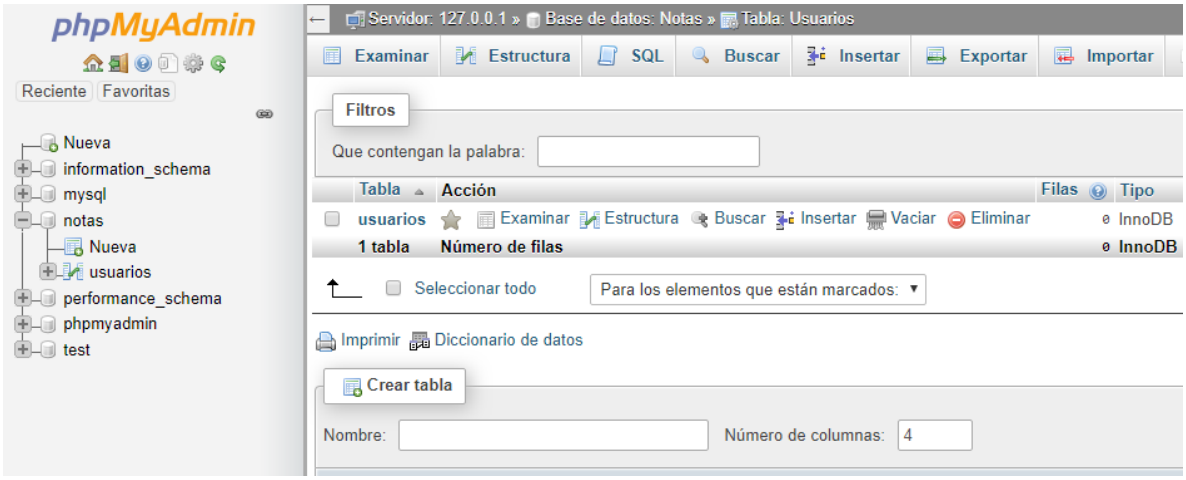
Usuarios				
Nombre	**Tipo**	**Longitud**	**Indice**	**Auto Increment**
ID_USUARIO	INT	15	Primario	Si
NOMBRE	VARCHAR	60	-	-
APELLIDO	VARCHAR	60	-	-
USUARIO	VARCHAR	40	-	-
PASSWORD	VARCHAR	80	-	-
PERFIL	VARCHAR	30	-	-
ESTADO	VARCHAR	20	-	-

Int: Permite almacenar únicamente valores numéricos.

Varchar: Permite almacenar cadenas de texto.

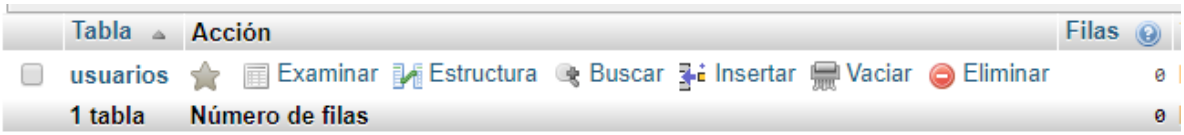


El resultado gráfico luego de guardar los respectivos cambios será el siguiente:



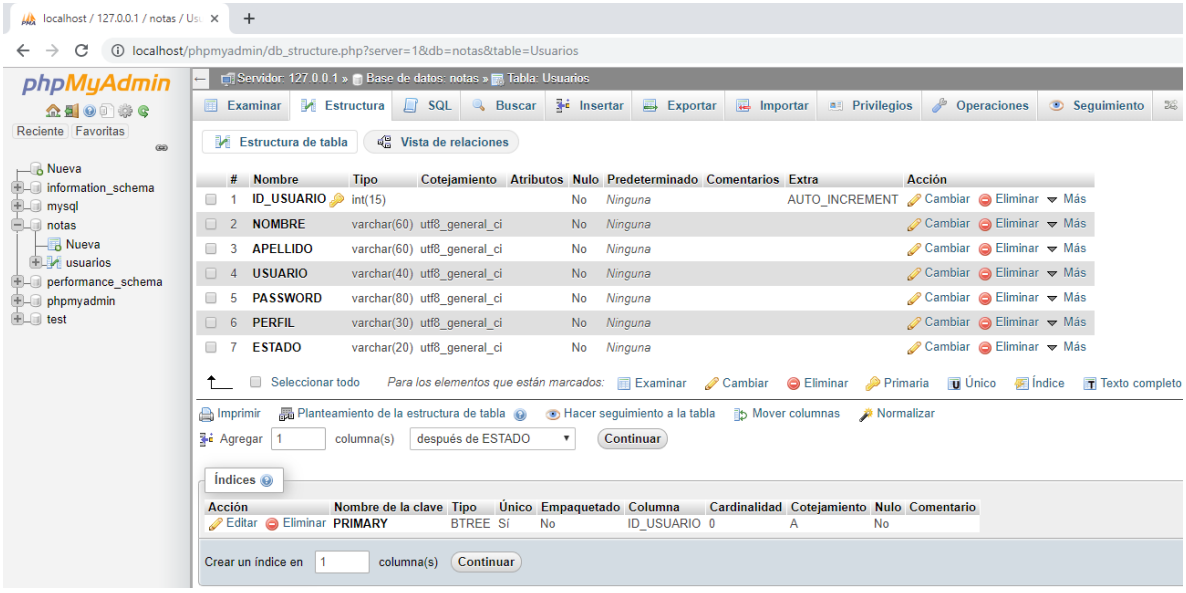
Ya contamos con nuestra primera tabla de nuestro proyecto con la estructura implementada.

Esta tabla ofrece ya ciertas funciones para operar sobre la misma, como lo son:



- Insertar datos.
- Examinar todos los datos.
- Buscar / filtrar.
- Vaciar todos los datos de la tabla.
- Eliminar la tabla.

Vayamos a la tabla y comencemos a operar sobre la misma:



Insertar datos

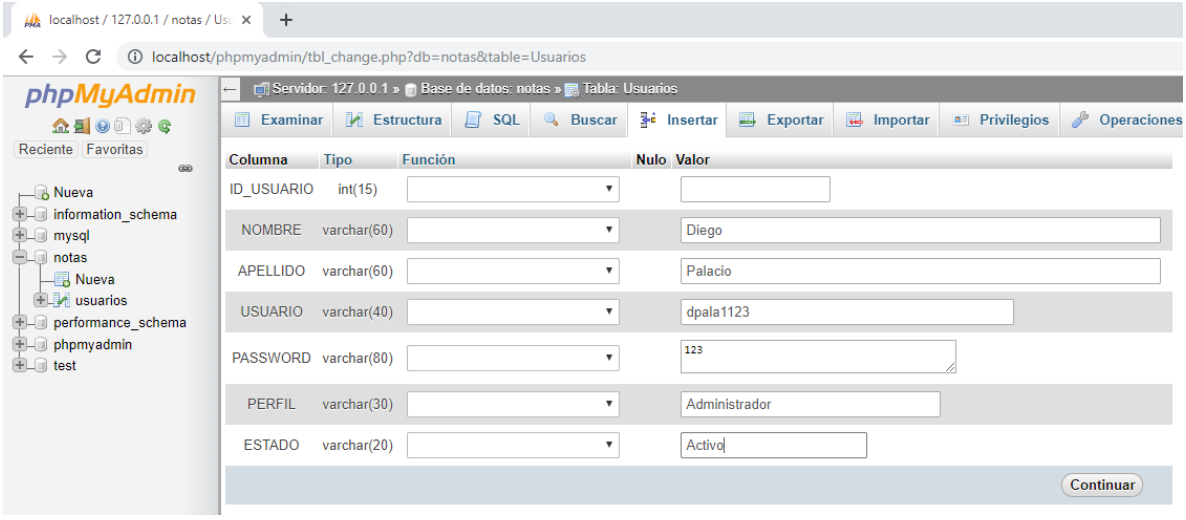
La primera función y la principal en manejo de tablas y de datos, es la inserción de los mismos, para insertar datos, lo podemos realizar de 3 formas:

- Por medio de la consola gráfica de phpMyAdmin.
- Por medio de la consola de código de phpMyAdmin.
- Importando un archivo de texto plano.

Para este caso utilizaremos las dos primeras opciones.

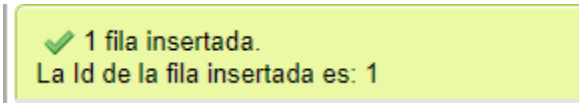
Consola gráfica

La consola gráfica ofrece un formulario con los respectivos campos determinados por la tabla en el apartado de "Insertar".



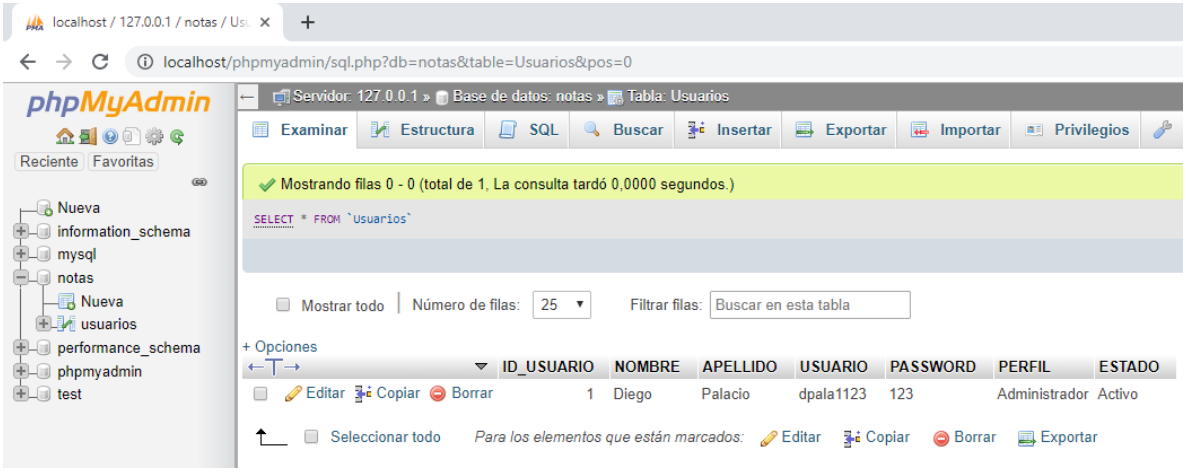
Los datos deben ser consistentes con los tipos y características solicitadas por la tabla.

El resultado de esta operación confirma la inserción exitosa de los datos y arroja un código de inserción:



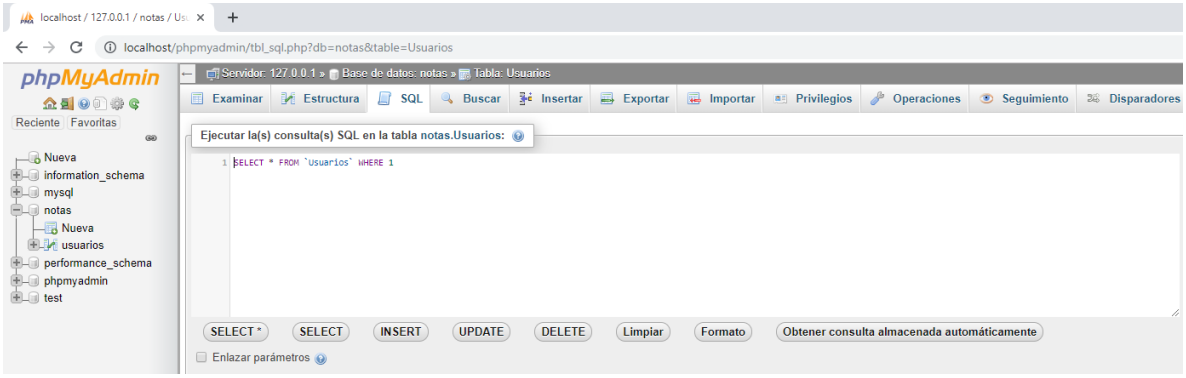
INSERT INTO `Usuarios` (`ID_USUARIO`, `NOMBRE`, `APELLIDO`, `USUARIO`, `PASSWORD`, `PERFIL`, `ESTADO`) VALUES (NULL, 'Diego', 'Palacio', 'dpala1123', '123', 'Administrador', 'Activo');

El resultado de la inserción puede observarse en el apartado de "Examinar" para visualizar todo el contenido.



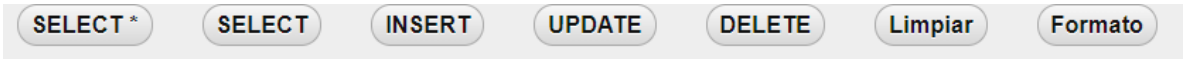
Consola de Código

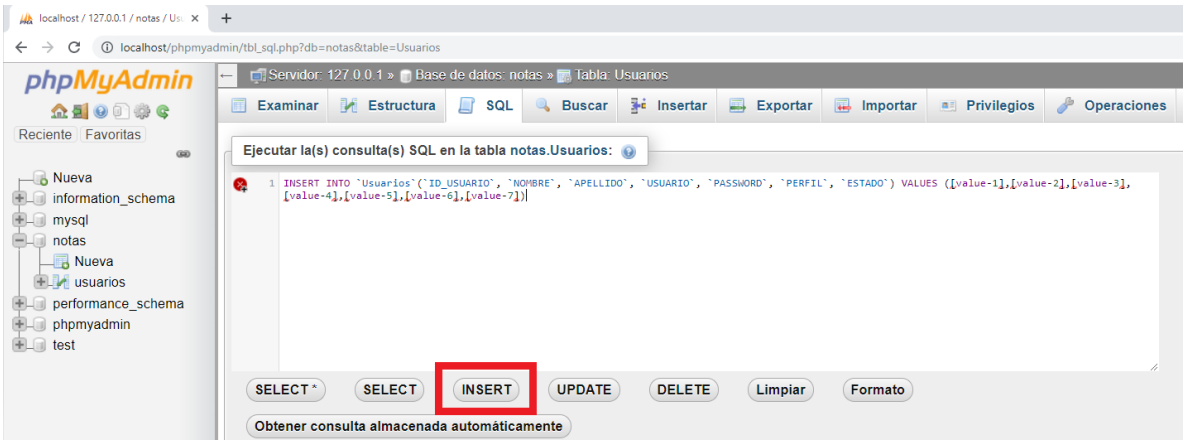
La consola de código ofrece una interfaz para realizar directamente consultas a una base de datos y más directamente sobre las tablas.



En esta interfaz es válido y aceptable cualquier tipo de consulta de MySQL, desde insertar, hasta eliminar. En este caso realicemos una inserción.

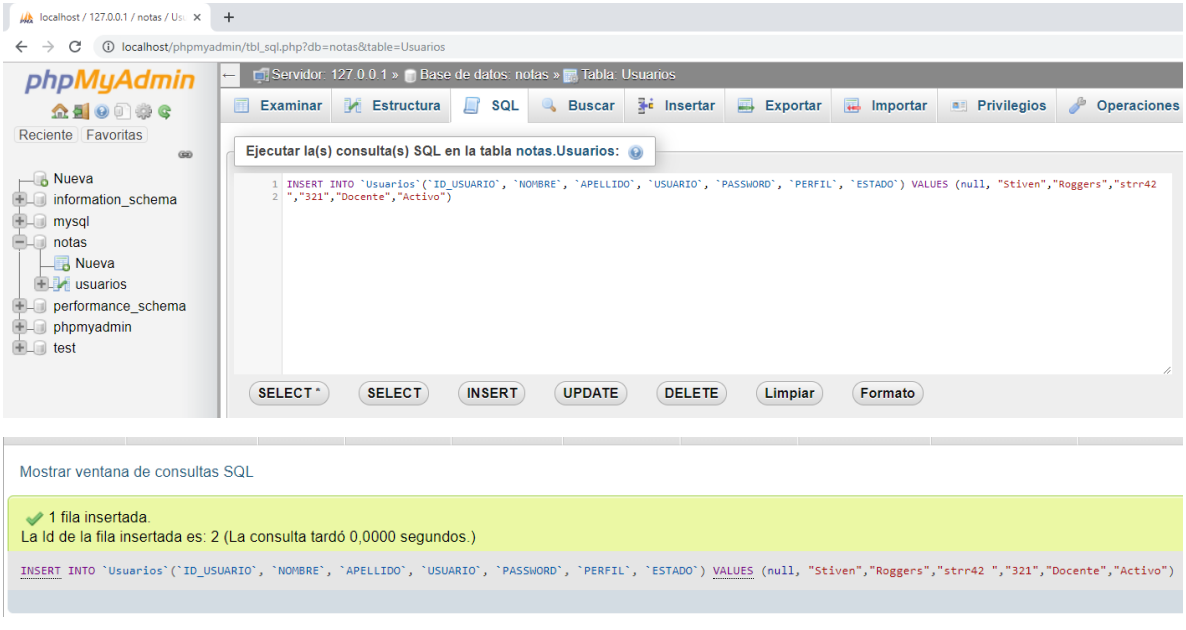
Entre las opciones predeterminadas de la intefaz, para este caso, usaremos “Insert”.





Estas opciones predeterminadas por la interfaz generan un código para realizar la operación deseada. En este caso un insert. Realicemos los respectivos cambios en este código.

INSERT INTO `Usuarios`(`ID_USUARIO`, `NOMBRE`, `APELLIDO`, `USUARIO`, `PASSWORD`, `PERFIL`, `ESTADO`) VALUES (null, "Stiven","Roggers","strr42","321","Docente","Activo")



Por medio de la instrucción Control + Enter podremos ejecutar el código o en la opción de "Continuar".



El resultado ahora serán dos datos insertados de dos formas diferentes, igual de funcionales, igual de prácticas. Recomiendo realizar esto casos por medio del código para practicar.

Tabla Materias

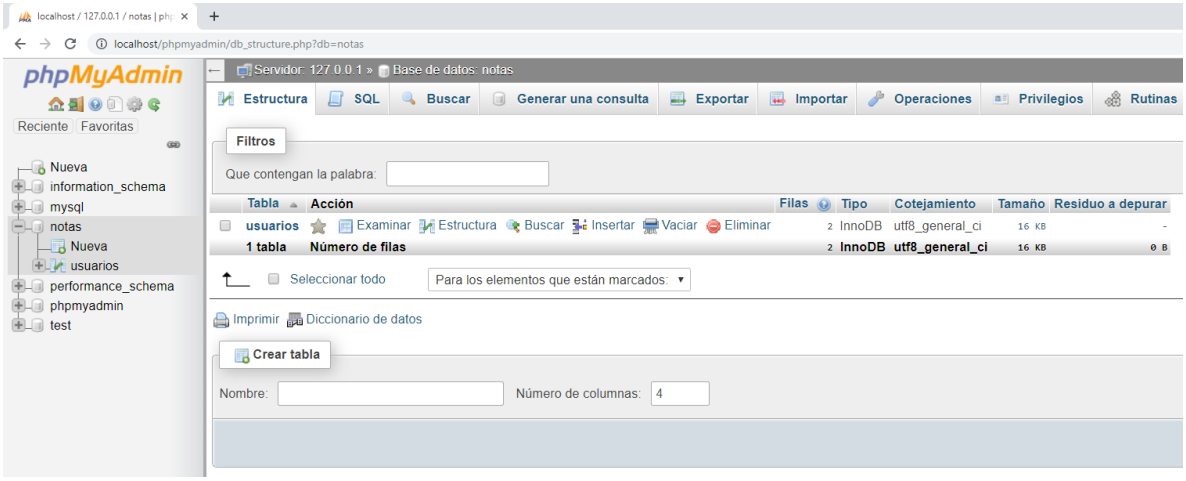
La tabla Materias dentro de nuestro sistema, contendrá la información de las respectivas materias que se evaluaran en el sistema. La información que contendrá nuestra tabla será:

Materias	
ID_MATERIA	**MATERIA**
1	Matematicas
2	Ciencias
3	Fisica
4	Ingles

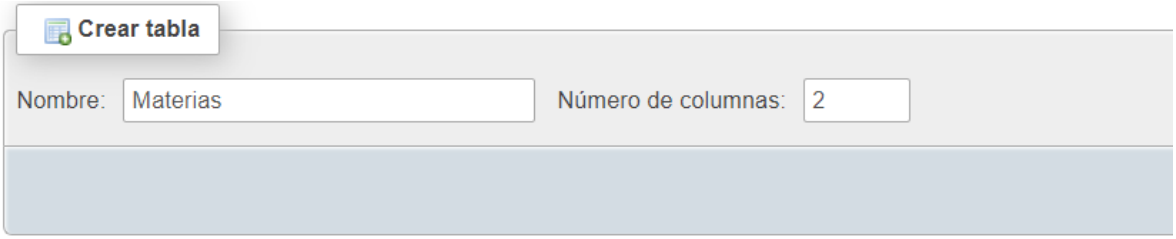
La tabla contará con la siguientes características e información para su correcta implementación.

- El nombre de la misma será Materias, el nombre es claro y diciente de la información que contendrá.
- Tiene 2 campos, con nombres claros y en singular.
- Cada campo deberá contar con una tipificación. Algo similar a PHP con los tipos de datos.
- Cada campo deberá contener un tamaño máximo de almacenamiento (caracteres).
- El primer campo por estándar para el diplomado será el ID y a su vez clave primaria (Un campo único en la tabla que permitirá reconocer y diferencia todos los registros. Ej: El Id 1, pertenece a la materia de matemáticas y nunca habrá otro Id 1).
- Otra característica del ID es que será auto_increment, es decir, cada vez que se registre un nuevo registro (valga la redundancia) el campo ID aumentará en razón de 1.

Para realizar la creación de la tabla “Materias” dentro de la base de datos “notas”, regresemos al apartado de phpMyAdmin principal de nuestra base de datos:

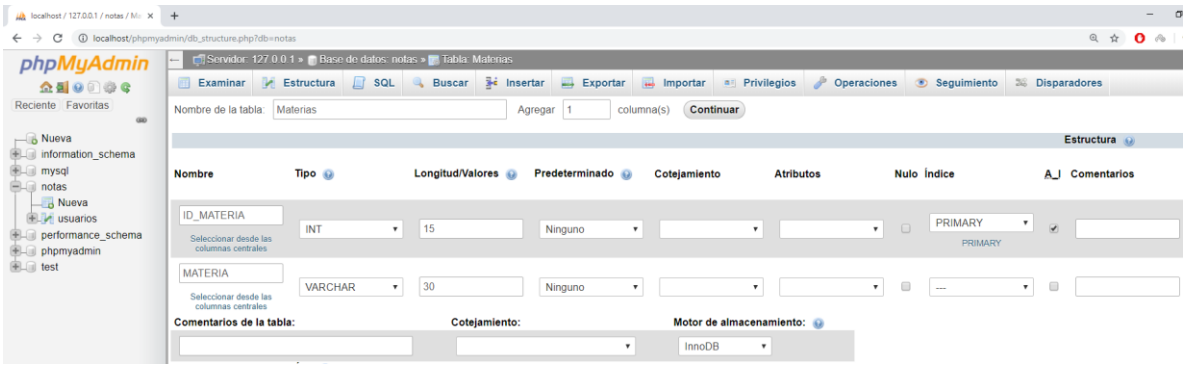


Simplemente de nuevo, asignamos nombre y número de columnas:



Definiendo las características de nuestra tabla, serían las siguientes:

Materias				
Nombre	**Tipo**	**Longitud**	**Indice**	**Auto Increment**
ID_MATERIA	INT	15	Primario	Si
MATERIA	VARCHAR	30	-	-



El resultado gráfico luego de guardar los respectivos cambios será el siguiente:



Realiza la inserción de las 4 respectivas materias de prueba adjuntadas en la guía: Matemáticas, Ciencias, Física e Inglés, con sus respectivos Id's.

Tabla Estudiantes

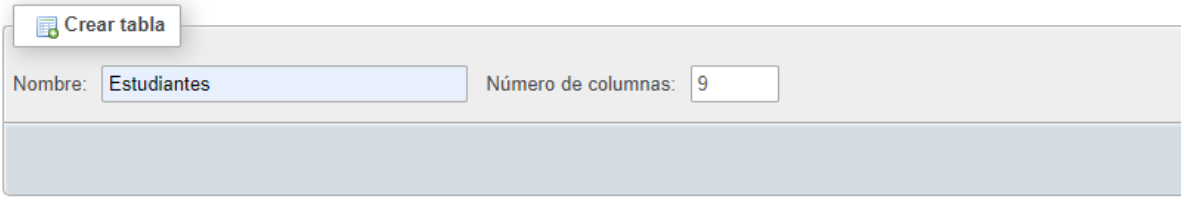
La tabla Estudiantes dentro de nuestro sistema, contendrá toda la información de los estudiantes del sistema. La información que contendrá nuestra tabla será:

Estudiantes								
ID_ESTUDIANTE	**NOMBRE**	**APELLIDO**	**DOCUMENTO**	**CORREO**	**MATERIA**	**DOCENTE**	**PROMEDIO**	**FECHA_REGISTRO**
1	Estefania	Lopez	1004321424	estefi@gmail.com	Ingles	Stiven	90	2019-10-23
2	Evelin	Gutierrez	4314216	evegu@gmail.com	Fisica	Diego	80	2019-08-02

La tabla contará con la siguientes características e información para su correcta implementación.

- El nombre de la misma será Estudiantes, el nombre es claro y diciente de la información que contendrá.
- Tiene 9 campos, con nombres claros y en singular.
- Cada campo deberá contar con una tipificación. Algo similar a PHP con los tipos de datos.
- Cada campo deberá contener un tamaño máximo de almacenamiento (caracteres).
- El primer campo por estándar para el diplomado será el ID y a su vez clave primaria (Un campo único en la tabla que permitirá reconocer y diferencia todos los registros. Ej: El Id 1, pertenece a la estudiante Estefania y nunca habrá otro Id 1).
- Otra característica del ID es que será auto_increment, es decir, cada vez que se registre un nuevo registro (valga la redundancia) el campo ID aumentará en razón de 1.

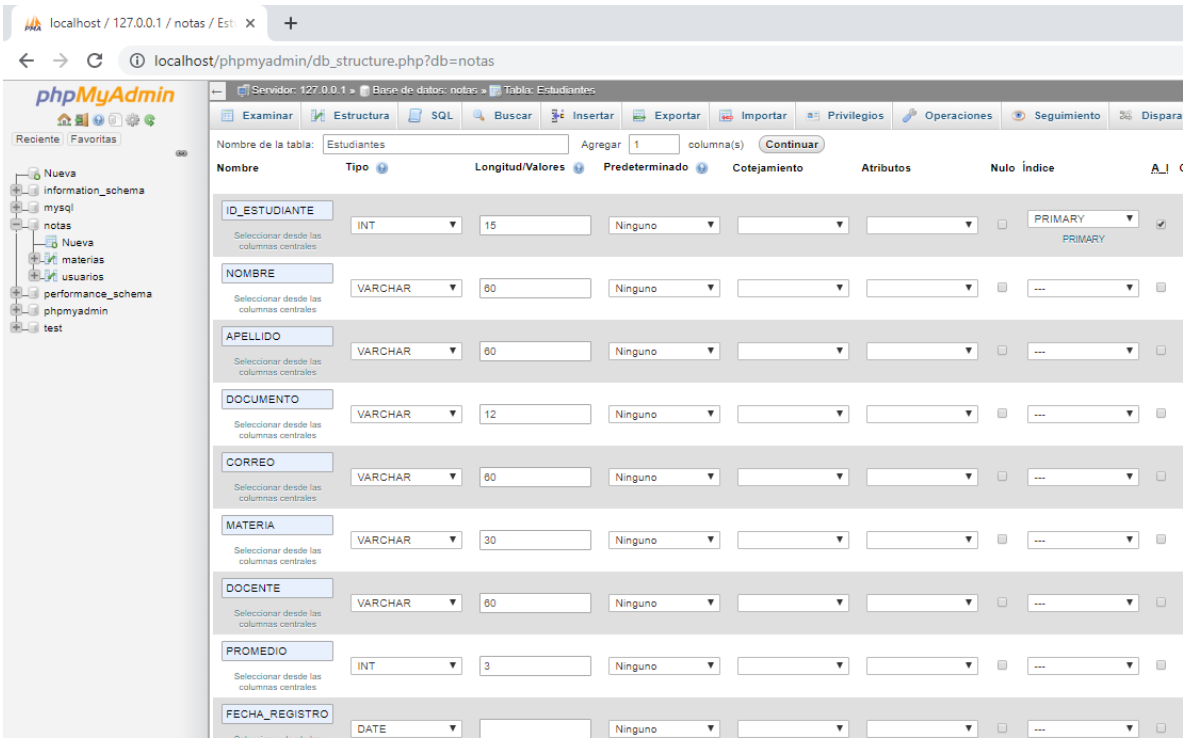
Para realizar la creación de la tabla "Estudiantes" dentro de la base de datos "notas", regresemos al apartado de phpMyAdmin principal de nuestra base de datos:



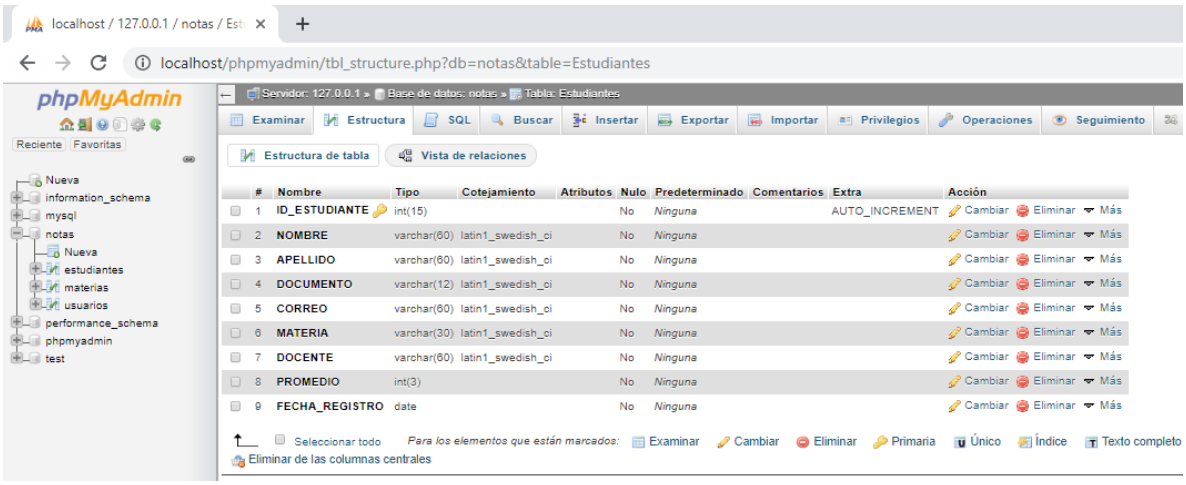
Definiendo las características de nuestra tabla, serían las siguientes:

Estudiantes				
Nombre	**Tipo**	**Longitud**	**Indice**	**Auto Increment**
ID_ESTUDIANTE	INT	15	Primario	Si
NOMBRE	VARCHAR	60	-	-
APELLIDO	VARCHAR	60	-	-
DOCUMENTO	VARCHAR	12	-	-
CORREO	VARCHAR	60	-	-
MATERIA	VARCHAR	30	-	-
DOCENTE	VARCHAR	60	-	-
PROMEDIO	INT	3	-	-
FECHA_REGISTRO	DATE	-	-	-

Date: Permite almacenar fechas en formato: YYYY-MM-DD.



El resultado gráfico luego de guardar los respectivos cambios será el siguiente:



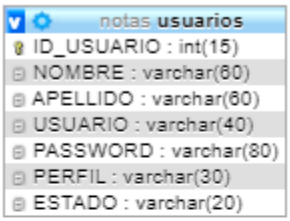
Realiza la inserción de los 2 respectivos estudiantes de prueba adjuntadas en la guía: Estefania y Evelin, con sus respectivos Id's.

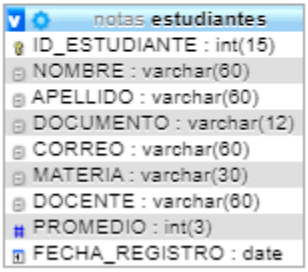
A este punto del proceso, debemos contar ya con nuestra base de datos completa, las 3 tablas declaradas y cada una de estas con los respectivos datos de prueba insertados.

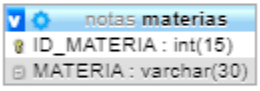
En el apartado de nuestra base de datos "notas" en las opciones de la parte superior está la opción de "Diseñador":



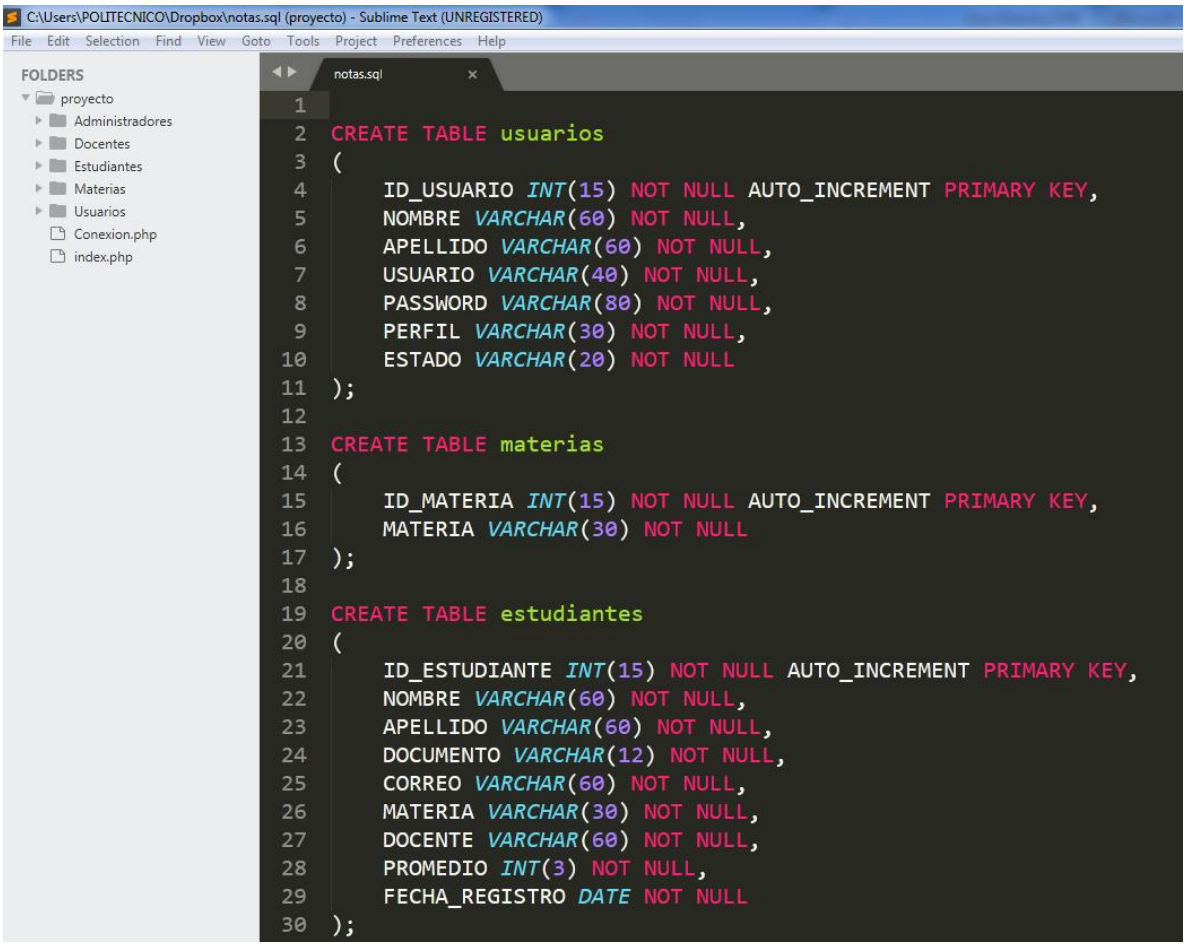
Esta opción nos permite obtener un esquema gráfico de cómo está estructurada nuestra base de datos, cómo están nuestras tablas, qué campos tiene la misma, de qué tipo son y cuál es el tamaño.



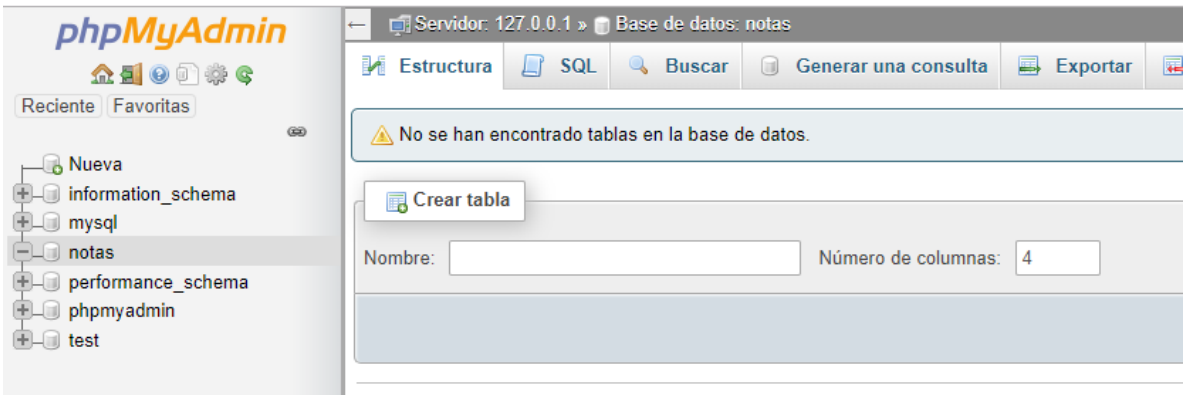




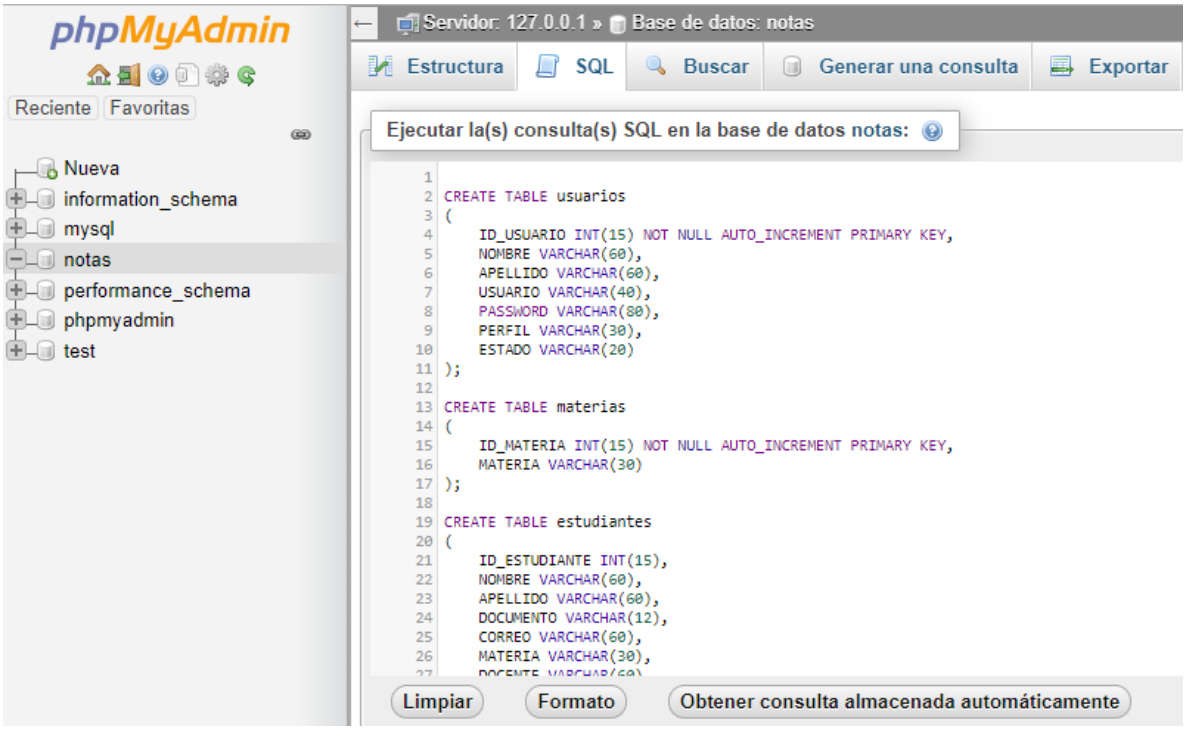
En los procesos anteriores hemos creado nuestras bases de datos y tablas de forma gráfica. Pero podemos realizar este proceso de igual forma por código:



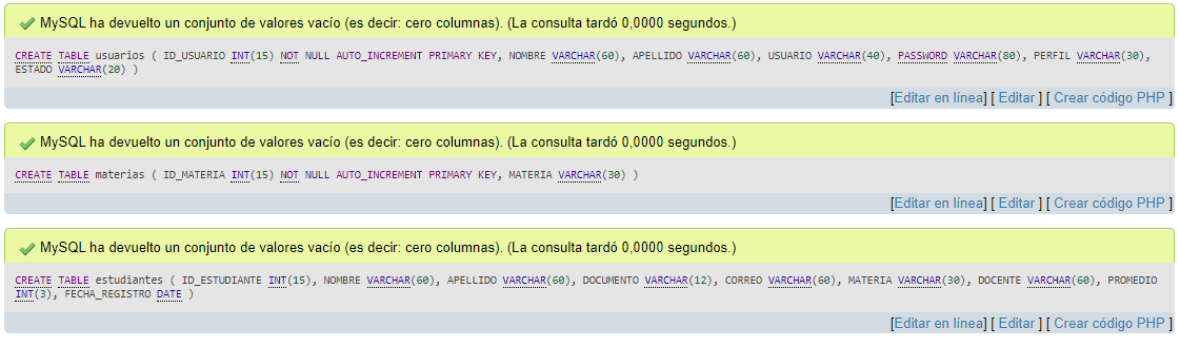
Para realizar este ejercicio, elimina todas las tablas creadas anteriormente y obtener el resultado inicial:



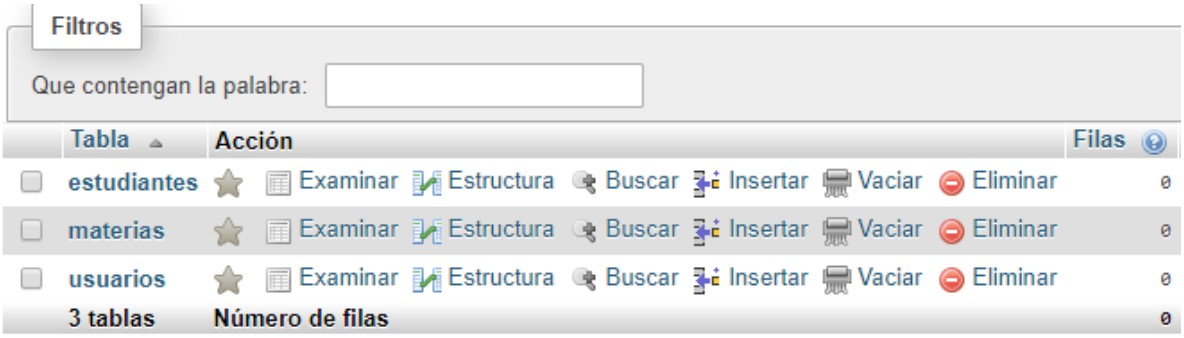
En el apartado de SQL de nuestra base de datos realizaremos la creación de todas las tablas de una forma "más rápida".



Transcribiendo directamente el código y ejecutando el mismo se obtendrá el siguiente resultado:



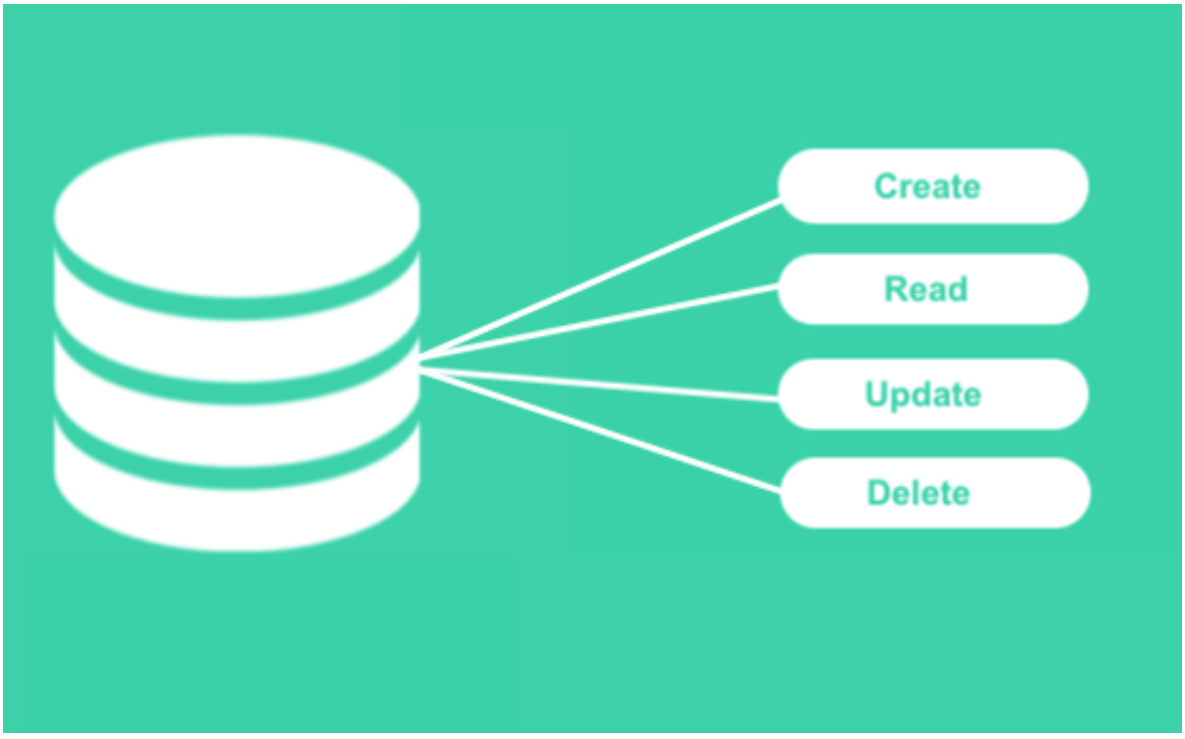
La creación exitosa de las tablas de nuestra base de datos:



TEMA 2
PHP - PDO

Volvamos hablar de PHP la programación, ahora hablado de PHP y de Bases de Datos es decir PDO.

PDO significa **PHP Data Objects**, **Objetos de Datos de PHP**, es una extensión para acceder a bases de datos. PDO permite acceder a diferentes sistemas de bases de datos con un controlador específico (MySQL, SQLite, Oracle...) mediante el cual se conecta. Independientemente del sistema utilizado, se emplearán siempre los mismos métodos, lo que hace que cambiar de uno a otro resulte más sencillo.



El sistema PDO se fundamenta en 3 clases: PDO, PDOStatement y PDOException.

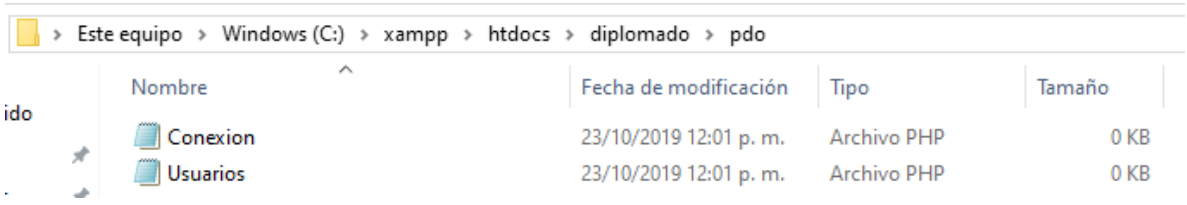
- La clase PDO se encarga de mantener la conexión a la base de datos, además de crear instancias de la clase PDOStatement.

- PDOStatement, se encarga de manejar las sentencias SQL y devuelve los resultados.

- La clase PDOException se utiliza para manejar los errores.

Bajo estas tres clases se fundamenta el desarrollo y procesos de bases de datos en PHP, veamos algunos ejemplos de inserción, consulta, actualización y eliminación de datos.

Dentro de nuestro folder diplomados, crearemos uno nuevo para realizar todas las pruebas y funciones de las bases de datos llamado "pdo". Volvamos a la programación orientada a objetos:

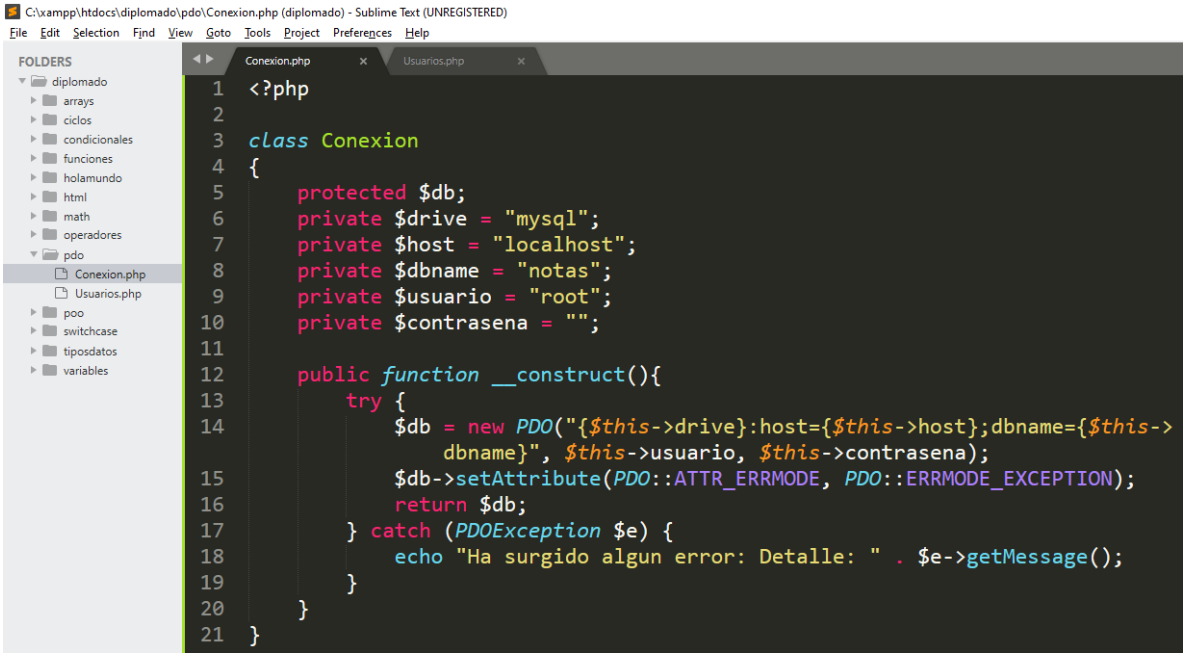


Esta será la estructura, dos clases dentro del folder de "pdo"

- Conexión: contendrá una clase con toda la conexión de nuestra base de datos.

- Usuarios: contendrá todos los procesos a realizar sobre la base de datos y en especial, nuestra tabla usuarios creada anteriormente.

Clase Conexión:

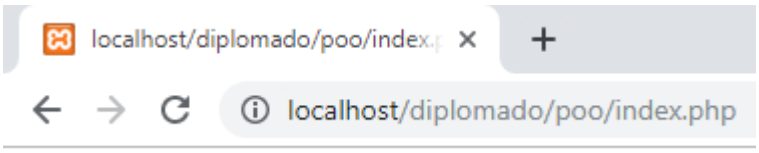


No te alarmes, veamos línea por línea lo que acaba de ocurrir en nuestra clase:

Nuestra clase inicia con la declaración de los respectivos atributos que contendrá:

```
protected $db;
private $drive = "mysql";
private $host = "localhost";
private $dbname = "notas";
private $usuario = "root";
private $contrasena = "";
```

- **Protected $db:** Determina la variable que contendrá el objeto de PDO para usar.
- **Private $drive = "mysql":** Especifica al driver el motor de base de datos a utilizar, en nuestro caso será mysql, aunque puede ser: sql, postgress, etc.
- **Private $host = "localhost":** Especifica el lugar donde se ejecutará la conexión, lo mismo ocurre con el lugar donde ejecutamos PHP:



- **Private $dbname = "notas":** Especifica el nombre (EXACTO) de nuestra base de datos (previamente creada):
- **Private $usuario = "root":** Especifica el usuario de conexión de nuestra base de datos, que por defecto al instalar XAMPP es "root".
- **Private $contrasena = "":** Especifica la contraseña de conexión de nuestra base de datos, que por defecto al instalar XAMPP es "", es decir, ninguna.

Estos son todos los atributos establecidos para generar una exitosa conexión por medio de MySQL, el siguiente paso/pieza de código será nuestro constructor:

```php
public function __construct(){
    try {
        $db = new PDO("{$this->drive}:host={$this->host};dbname={$this->
            dbname}", $this->usuario, $this->contrasena);
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $db;
    } catch (PDOException $e) {
        echo "Ha surgido algun error: Detalle: " . $e->getMessage();
    }
}
```

- Primero que todo, qué es "**try {} catch () {}",** estas líneas de código señalan un bloque de instrucciones a intentar (try), y especifica una respuesta si se produce una excepción (catch). Es decir, en nuestro caso try catch, será el contener que se encargará de realizar los procesos en caso de que todo el código se encuentre bien o arrojará un error (Exception) en caso de que no lo sea.

- $db = new PDO(….): Crea un instancia de la clase PDO, que en su constructor recibe todos los datos para realizar una conexión:

  a. $dns: El Nombre del Origen de Datos (DSN), contiene la información requerida para conectarse a la base de datos. En nuestro caso la información contenida entre llaves "{}" y a su vez contiene: Drive – Host – Nombre de la base de datos.

  b. $usuario.

  c. $contrasena.

En caso de que nuestra instancia ($db) y los parámetros que recibe sean los correos y todo esté bien, devolverá un objeto exitosamente y si no, se lanzará una exception.

- $db->setAttribute(…): PDO maneja los errores en forma de excepciones, por lo que la conexión siempre ha de ir encerrada en un bloque try/catch. Se puede (y se debe) especificar el modo de error estableciendo el atributo error mode:
  a. PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION.

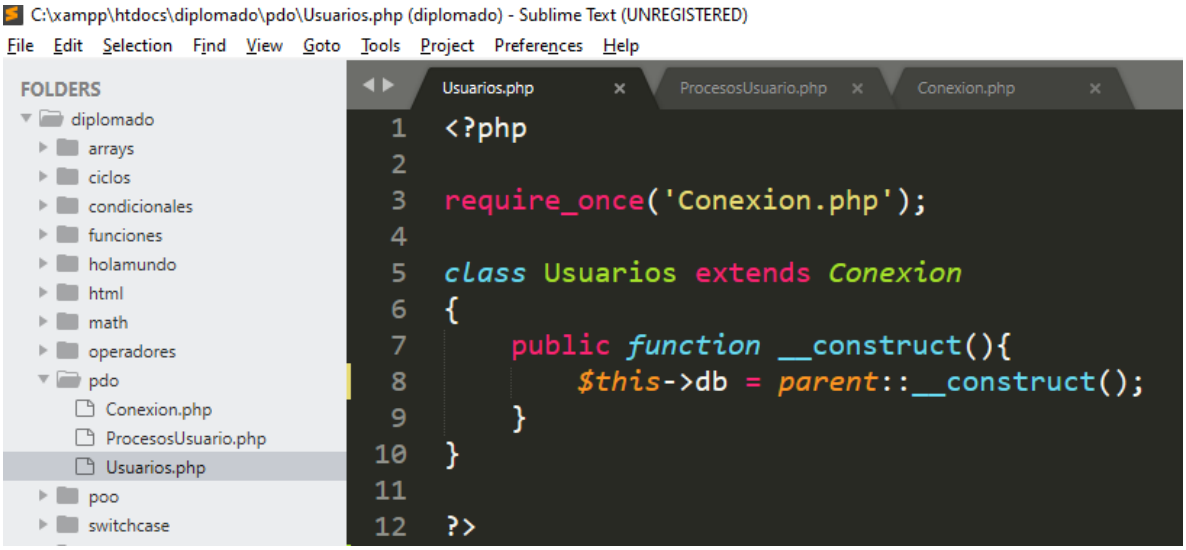
  No importa el modo de error, si existe un fallo en la conexión siempre producirá una excepción, por eso siempre se conecta con try/catch.

- Finalmente, en caso de que todo esté bien. Simplemente se retornará el objeto: return $db.

- catch (PDOException $e): Se encarga de crear la exceptión y detener todo el proceso.

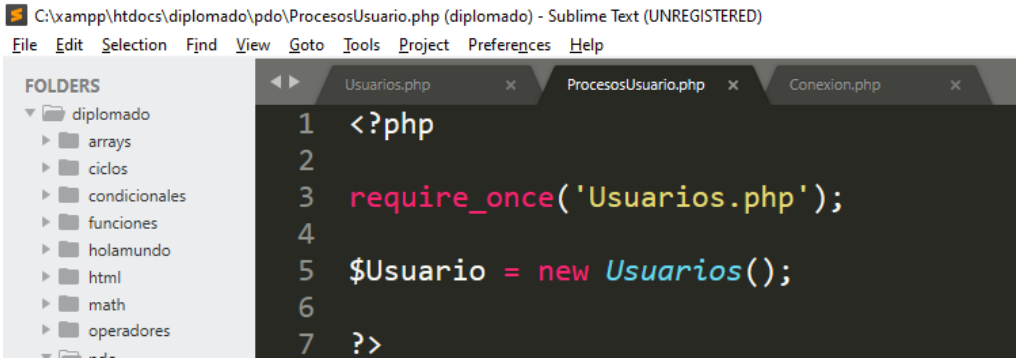
Este es todo el código que se ocupa para realizar una conexión en PHP, para realizar uso de esta, es necesario utilizar herencia, volvamos a la programación orientada a objetos y sus buenas prácticas.

Ahora implementemos la clase Usuarios para realizar una prueba de la conexión, para conocer un resultado exitoso o erróneo.



La clase Usuarios simplemente realiza una herencia de la Conexión y el constructor de la clase hijo, Usuario realiza un llamado al constructor de la clase Padre, Conexión. Este resultado, que en realidad recordando a lección anterior, retorna un objeto de la clase PDO, será almacenada en la variable $db para hacer uso de la misma. Es decir, por medio de la herencia haremos uso del atributo db que la clase conexión nos retorna en caso de obtener una conexión exitosa.

Al igual que en la programación orientada a objetos, para realizar una operación sobre las clases, ocupamos un nuevo archivo para realizar los llamados a la misma, en este caso lo llamaremos: "ProcesosUsuario.php" y contendrá simplemente creará un objeto.



Para esta prueba, realicemos el siguiente cambio en la clase Conexión y en su constructor:

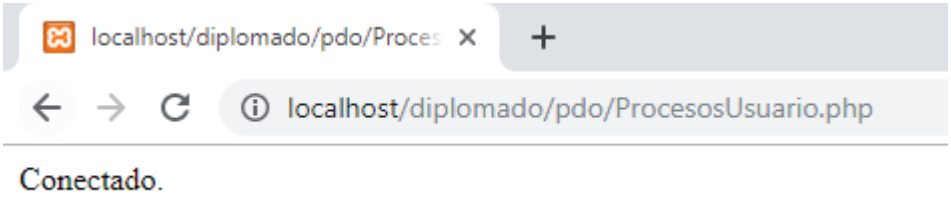
```php
public function __construct(){
    try {
        $db = new PDO("{$this->drive}:host={$this->host};dbname={$this->
            dbname}", $this->usuario, $this->contrasena);
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        echo "Conectado.";

        return $db;
    } catch (PDOException $e) {
        echo "Ha surgido algun error: Detalle: " . $e->getMessage();
    }
}
```

Un simple echo "Conectado.".

El ir al navegador al archivo "ProcesosUsuario.php": http://localhost/diplomado/pdo/ProcesosUsuario.php



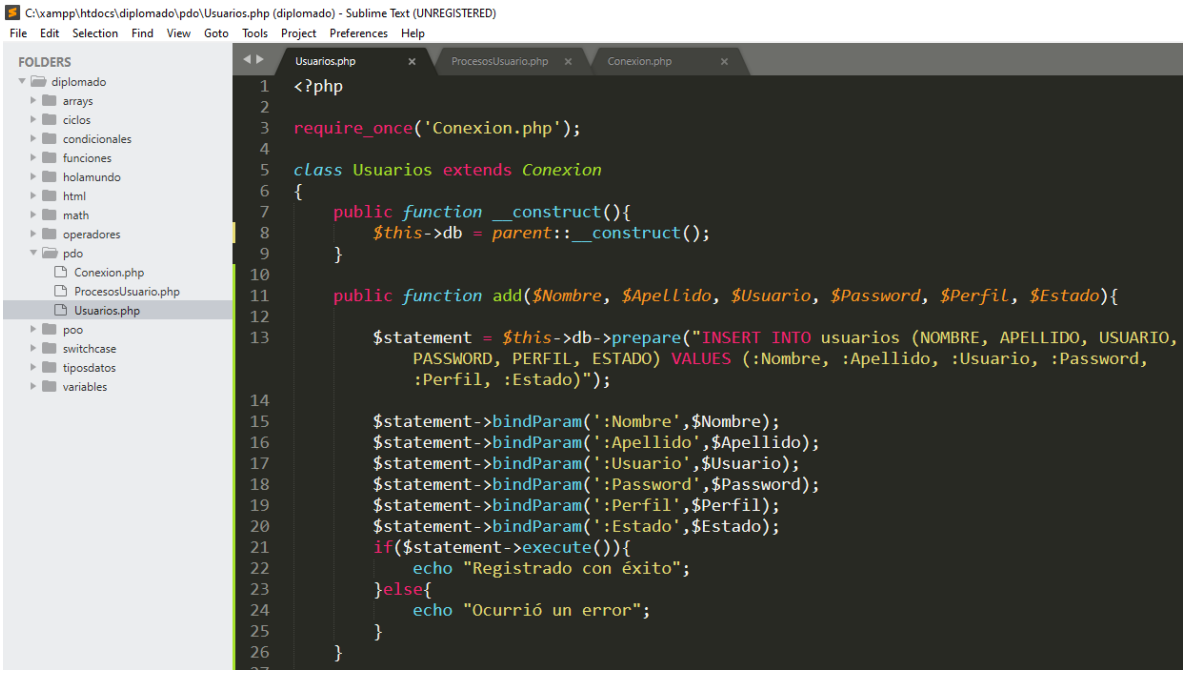
Estaremos conectados exitosamente en nuestro base de datos. Ahora podremos realizar operaciones sobre la base de datos. Veamos:

Insertar

La función insertar es muy sencilla, consta de un método que realizará uso de las clases de PDO para operar sobre las tablas de nuestra base de datos. Sintaxis:

INSERT INTO nombre_tabla (columna1, columna2, …) VALUES (:value1, :value2, …)

Veamos el código.



En primera instancia, recordando la programación orientada a objetos, los métodos deben ser muy bien declarados, en este caso, todos deben de ser públicos, contar con la palabra reservada function y el nombre del método (que puede ser cualquiera, siempre y cuando sea descriptivo. Ej: add, agregar, insertar, registrar. Etc).

El método debe recibir los parámetros que se desean insertar en la sentencia de SQL. Es decir, debemos basarnos en los descriptos en nuestra tabla de la base de datos.

ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO

Entonces, según esto, debemos pasar un parámetro por cada columna de la tabla. Exceptuando el ID, dado que especificamos que fuese automático. Para nuestro caso serían entonces: $Nombre, $Apellido, $Usuario, $Password, $Perfil, $Estado.

Teniendo nuestros parámetros, podemos proceder a realizar nuestra instrucción para insertar los datos. Recuerda que al insertar datos en XAMPP, se tenían 2 opciones, por código o de forma gráfica. Por este medio realizaremos únicamente instrucciones por medio de código.

$statement = $this->db->prepare("INSERT INTO usuarios (NOMBRE, APELLIDO, USUARIO, PASSWORD, PERFIL, ESTADO) VALUES (:Nombre, :Apellido, :Usuario, :Password, :Perfil, :Estado)");

$statement, será la variable que contendrá el objeto de la sentencia de MySQL generada por el **prepare**. Para preparar nuestra sentencia, debemos hacer uso del objeto que extendemos de nuestra clase conexión ($this->db->prerpare) para poder acceder al método prepare.

Prepare(..), Este método prepara una sentencia para su ejecución y devuelve un objeto sentencia. El parámetro de este método recibirá una sentencia MySQL, en este caso un INSERT.

Nuestro INSERT se compone de las siguientes características que lo denotan.

a. INSERT INTO: son las palabras reservadas de MySQL para indicar que se realizará una inserción.

b. "usuarios": especifica la base de datos sobre la cual se desea realizar la operación determinada anteriormente.

c. (NOMBRE, APELLIDO…..): hace referencia a los campos de la base de datos en el orden que se encuentran referenciados en nuestra tabla previamente especificada.

d. VALUES: es una palabra reservada de MySQL que indica que se realizará la referencia de los valores a partir de los campos de la base de datos.

e. (:Nombre, :Apellido….): representa los parámetros recibidos por el método que presentan cada uno de los campos de la base de datos en orden. Es decir, en el punto (**c**) especificamos los campos sobre los cuales deseamos insertar datos, y en el punto (**e**) referenciamos ya los valores. Es muy importante, si referenciamos 6 campos de la tabla de nuestra base de datos, debemos referenciar 6 parámetros. Cada uno inicia con ":" y finaliza separando cada uno por ",". Ej: (:Nombre, :Apellido, :Usuario, :Password, :Perfil, :Estado).

El resultado final es este:

```
$statement = $this->db->prepare("INSERT INTO usuarios (NOMBRE, APELLIDO,
    USUARIO, PASSWORD, PERFIL, ESTADO) VALUES (:Nombre, :Apellido,
    :Usuario, :Password, :Perfil, :Estado)");
```

Obteniendo de esta forma en el $statement, el objeto de la clase PDOStatemet, resultado de la consulta preparada, podemos ahora, "setear" los valores a nuestra consulta veamos.

Nuestro método recibe 6 parámetros, y nuestra sentencia preparada cuenta con esos mismos parámetros "referenciados", para poder asignar los valores finalmente a nuestra sentencia, debemos hacer uso del método **bindParam.**

BindParam se encarga de vincular un parámetro al nombre de variable especificado, un ejemplo sería el parámetro de "$Nombre "recibido por el método y la variable establecida en la sentencia preparada ":Nombre". El proceso se realiza de la siguiente forma:

```
$statement->bindParam(':Nombre',$Nombre);
```

Debemos hacer uso principalmente del objeto creado de la clase **PDOStatement** para acceder a **binParam**, este método recibe:

- La variable referenciada en la sentencia preparada.
- El parámetro que contiene el valor que se asignará a la sentencia.

Para nuestro caso el resultado para todas nuestras variables y nuestros parámetros sería este.

```
$statement->bindParam(':Nombre',$Nombre);
$statement->bindParam(':Apellido',$Apellido);
$statement->bindParam(':Usuario',$Usuario);
$statement->bindParam(':Password',$Password);
$statement->bindParam(':Perfil',$Perfil);
$statement->bindParam(':Estado',$Estado);
```

Según esto se debe crear un **bindParam** para cada parámetro que deseamos enlazar con nuestra sentencia preparada.

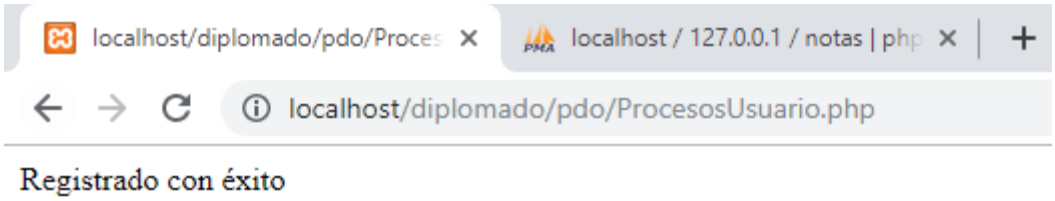
Finalmente debemos ejecutar nuestra sentencia preparada por medio del método **execute(),** este método se encarga de ejecutar nuestra sentencia prepara y las referencias de los valores, en caso de que todo salga bien, retornará true, sino, false.

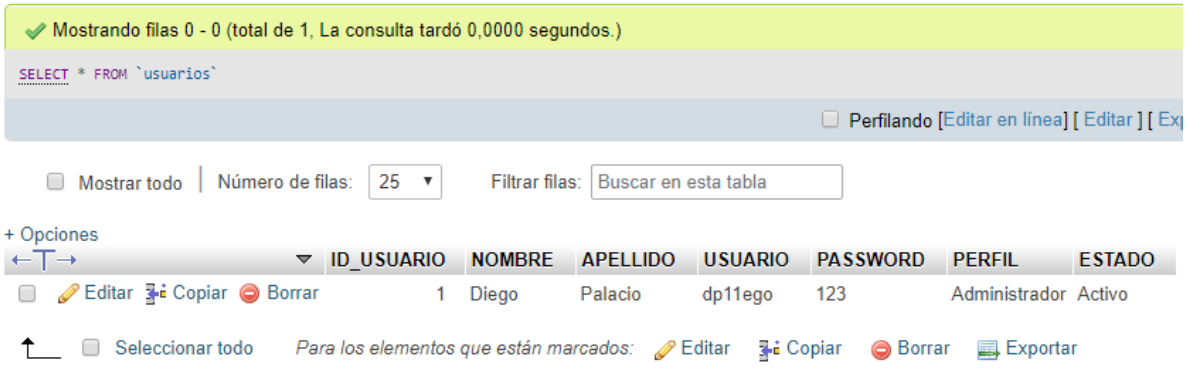
Por esto mismo se valida por medio de una condicional que el resultado sea exitoso o erróneo.

```php
if($statement->execute()){
    echo "Registrado con éxito";
}else{
    echo "Ocurrió un error";
}
```

Ahora para realizar pruebas, al igual que en la programación orientada a objetos, realicemos un llamado al método add de la clase Usuarios:





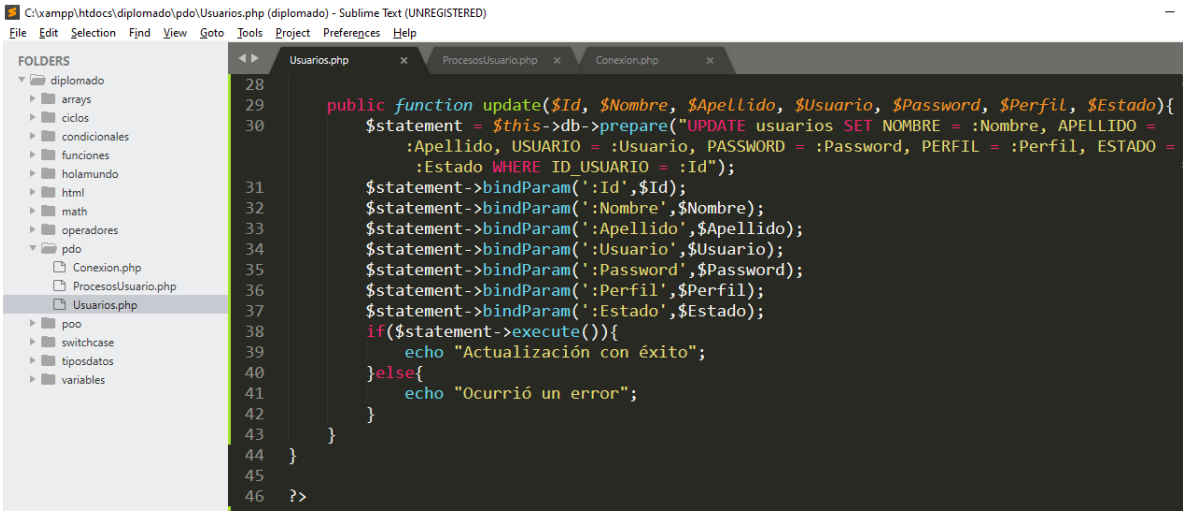

Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0000 segundos.)
SELECT * FROM `usuarios`
Perfilando [Editar en línea] [Editar] [Ex
Mostrar todo
Número de filas: 25
Filtrar filas: Buscar en esta tabla
+ Opciones
ID_USUARIO NOMBRE APELLIDO USUARIO PASSWORD PERFIL ESTADO
Editar Copiar Borrar 1 Diego Palacio dp11ego 123 Administrador Activo
Seleccionar todo
Para los elementos que están marcados:
Editar Copiar Borrar Exportar

Actualizar

La función actualizar es muy sencilla, parecido al insertar, consta de un método que realizará uso de las clases de PDO para operar sobre las tablas de nuestra base de datos. Sintaxis:

UPDATE nombre_tabla SET columna1 = :value1, columna2 = :value2 WHERE columna_condicion = :value_condicion

Veamos el código.



El código opera de la misma forma, lo único que cambia en este método es la sentencia preparada, para el insert hacíamos uso de la palabra reservada **INSERT** ahora haremos uso de **UPDATE**.

Omitiré la explicación de algunos conceptos y entraré en lo nuevo de este método.

La principal condición de los **UPDATEs** es que se realiza sobre registros ya insertados y que se encuentran en la base de datos. Cuando operamos con **UPDATEs**, hay una característica que se debe tener muy presente y es la palabra reservada **WHERE**.

WHERE es una cláusula de condición que permite para las sentencias **UPDATEs** determinar qué voy actualizar y qué no voy actualizar. Por ejemplo:

Si en la tabla usuarios quisiera cambiar el **ESTADO** de uno de los registros, debo agregar la cláusula **WHERE** y realizando la condición: Ej:

En el siguiente registro tenemos la información de toda una persona y tenemos un dato muy importante que asigna especialmente la misma tabla, y este es el **ID**.

	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
ar Copiar Borrar	1	Alejandro	Palacio	dp11ego	123	Administrador	Activo

Si sobre este registro quisiéramos actualizar el **ESTADO** del registro a Inactivo. Simplemente deberé agregar la siguiente condicional con el **WHERE**.

…… WHERE ID_USUARIO = 1

Con este simple bloque de código, determinamos que solo actualizaremos el registro que tenga un ID = 1, sino agregásemos una condicional, la sentencia afectaría toda la tabla.

Según lo anterior, 6 parámetros no será suficientes para realizar una actualización correcta de datos; ahora deberemos adjuntar el $Id del usuario:

$Id, $Nombre, $Apellido, $Usuario, $Password, $Perfil, $Estado

El ID será nuestro elemento clave para realizar condicionales a la hora de actualizar.

Por otra parte, encontramos la palabra reservada SET, se indica en la sentencia las columnas que será afectas o los valores que tendrán las mismas en el siguiente orden a partir de la sintaxis:

COLUMNA1 = :Valor1, COLUMNA2 = :Valor2

Las columnas en igual medida que en los insertar, al igual que los valores, el orden y la consistencia se deben mantener: NOMBRE con :Nombre, APELLIDO con :Apellido. En la sentencia se puede especificar únicamente que datos quiero actualizar; si sólo desease actualizar NOMBRE y APELLIDO, lo más acertado será únicamente agregar estos dos, pero para efectos prácticos de este ejercicio lo haremos todos. El resultado final de nuestra sentencia será:

```
$statement = $this->db->prepare("UPDATE usuarios SET NOMBRE = :Nombre, APELLIDO =
    :Apellido, USUARIO = :Usuario, PASSWORD = :Password, PERFIL = :Perfil, ESTADO =
     :Estado WHERE ID_USUARIO = :Id");
```

Una forma amigable de leer esta sentencia sería la siguiente: De la tabla usuarios, actualizar los siguientes campos, NOMBRE será igual a :Nombre, APELLIDO será igual a :Apellido, USUARIO será igual a :Usuario, PASSWORD será igual a :Password, PERFIL será igual a :Perfil y ESTADO será igual a :Estado, siempre y cuando se cumpla la condición de que el ID_USUARIO sea igual a :Id.

Finalmente realizamos la asignación de valores por medio de **bindParam**, adicionalmente es importante resaltar que el orden no importa, simplemente se debe cumplir con asignar correctamente los valores de los parámetros a las variables de la sentencia preparada.

```
$statement->bindParam(':Id',$Id);
$statement->bindParam(':Nombre',$Nombre);
$statement->bindParam(':Apellido',$Apellido);
$statement->bindParam(':Usuario',$Usuario);
$statement->bindParam(':Password',$Password);
$statement->bindParam(':Perfil',$Perfil);
$statement->bindParam(':Estado',$Estado);
```

Y ejecutamos nuestra sentencia preparada por medio de execute() contenida de nueva dentro de la condicional:

```
if($statement->execute()){
    echo "Actualización con éxito";
}else{
    echo "Ocurrió un error";
}
```

En este momento en nuestra tabla usuarios contamos con la siguiente información registrada:

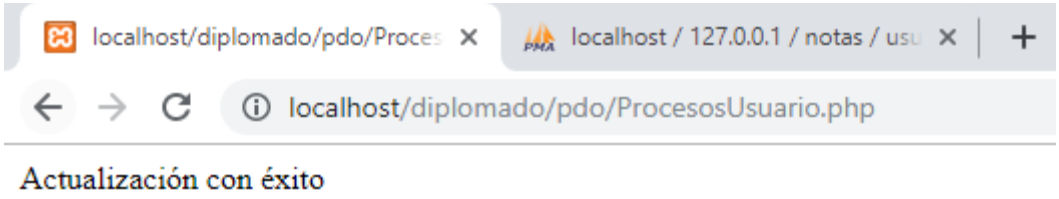
Opciones	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
Editar Copiar Borrar	1	Diego	Palacio	dp11ego	123	Administrador	Activo

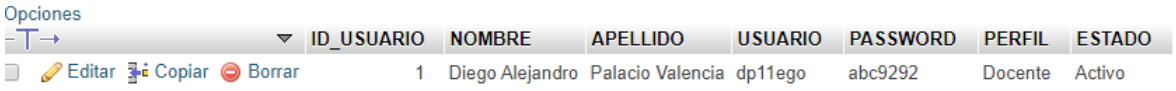
Hasta el momento debemos contar mínimamente con este registro, realicemos un **UPDATE** sobre este, cambiemos la siguiente información:

- El perfil pasará de ser Administrar a Docente.
- Agregaremos un Segundo Nombre (Diego Alejandro) y Segundo Apellido (Palacio Valencia).
- La nueva contraseña ahora será abc9292.

Los demás datos se deberán conservar tal como están:







El proceso funcionó correctamente.

Nota: no solamente puedes realizar condicionales con la cláusula **WHERE** y los **IDs**, puedes utilizar cualquier dato, fechas, nombres, edad, etc. Solo que, por convención, en la mayoría de los casos se usan los IDs.

Listar - Get

La función listar es muy sencilla, esta permite consultar datos de una o varias tablas. Consta de un método que realizará uso de las clases de PDO para operar sobre las tablas de nuestra base de datos. Sintaxis:

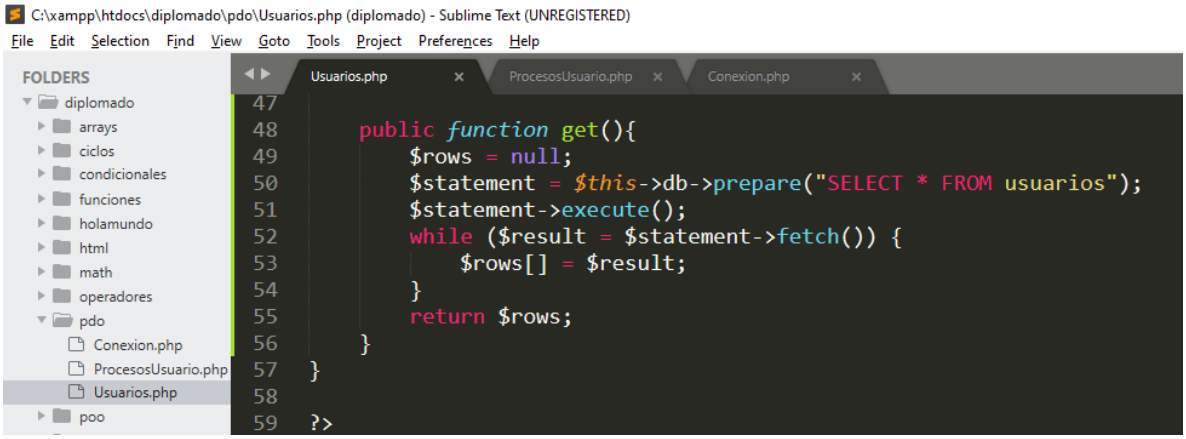
*SELECT * FROM nombre_tabla*

SELECT columna1, columna2… FROM nombre_tabla

Las dos sintaxis son completamente válidas, la primera que usa el (*) indica que traerá toda la información de los campos. La segunda, por otra parte, indica qué campos quiero traer separados por ",".

Veamos el código:

Sintaxis 1:



El código es muy corto y sencillo, la forma de operar es la siguiente:

El método no recibe parámetros, únicamente se recibe, en caso de que se desee implementar una clausula **WHERE** personalizada. Más adelante veremos esto.

La forma en que retornaremos los datos de nuestra tabla será por medio de una array, este lo declaramos de la siguiente forma:

$rows = null; vacío, dado que al final realizaremos la inserción de los datos en el mismo.

Nuestra sentencia preparada contiene una nueva palabra reservada **SELECT**: Select indica a la sentencia preparada que se realizará una selección de datos.

Al no recibir parámetros, no hace falta realizar referencia de valores por medio de bindParam, así que directamente se realiza la ejecución de la sentencia por medio de **execute().**

En las anteriores funciones, realizábamos el **execute()** y ahí finalizaba el proceso, dado que no ocupábamos realizar retornos de información, pero cuando trabajamos con sentencias de selección y consulta, es muy importante el retorno y para esto creamos el arreglo $rows (filas).

Dentro de los métodos que utilizamos en PDO hay uno muy especial y útil para trabajar con **SELECTs** y es el **fetch().**

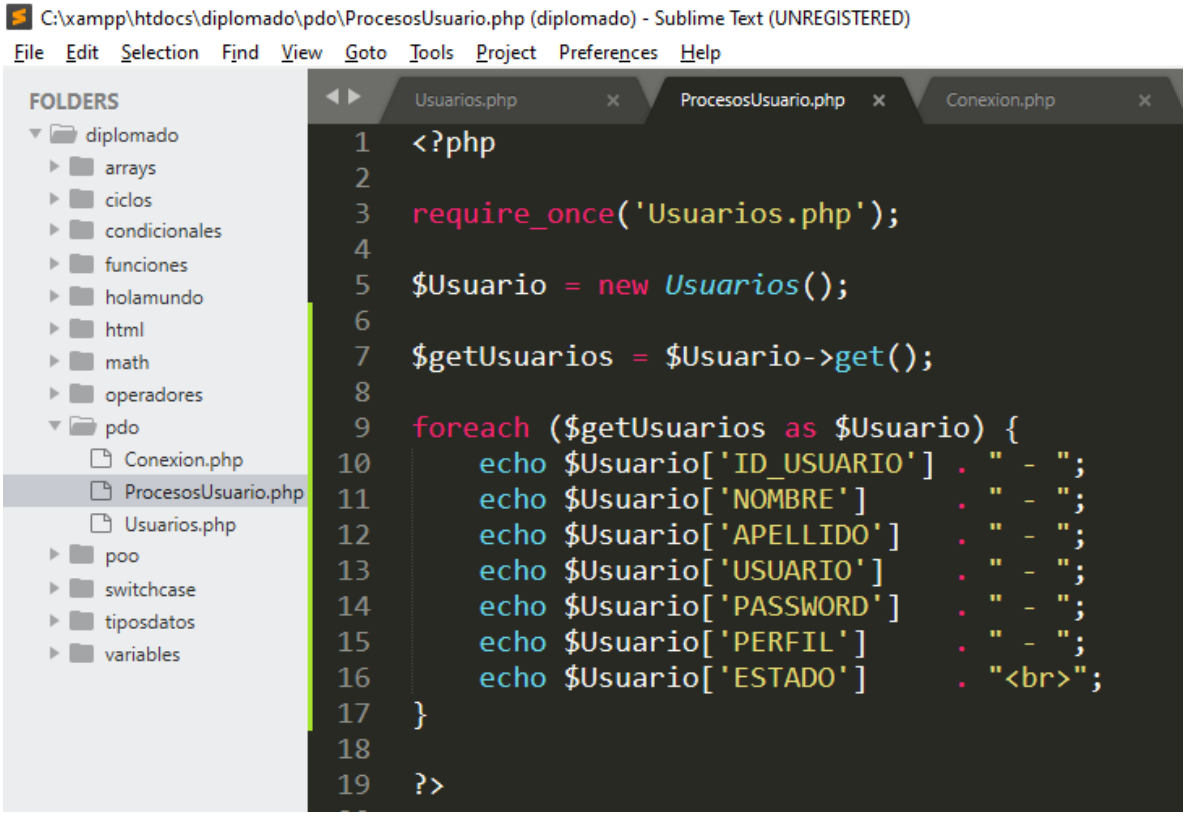
Fetch() permite recorrer sentencias de selección fila por fila hasta finalizar el recorrido. Pero este es un proceso repetitivo, o sea, un ciclo; recordemos que estructura se acomoda más a lo que necesitamos, ¿Ya lo tienes?, los ciclos while, ¿cierto?

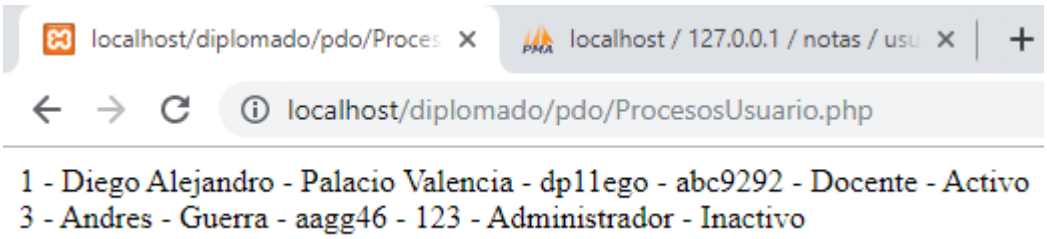
Los ciclos while recorrerán el resultado de **fetch()** hasta que no encuentre más iteraciones por realizar y esos resultados irán siendo guardados en nuestro arreglo de la siguiente forma:

```php
while ($result = $statement->fetch()) {
    $rows[] = $result;
}

return $rows;
```

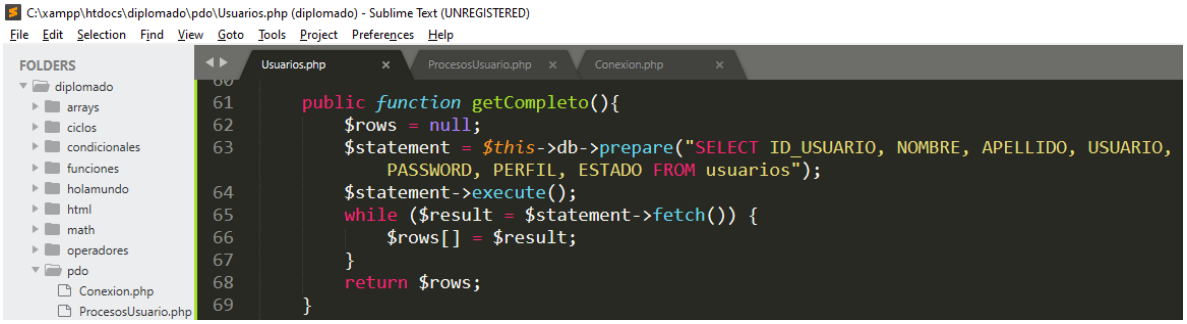
Result contendrá el resultado de UNA fila a la vez, recuerda que el ciclo while evalúa una condición y si es verdadera realiza el proceso interno. Al encontrar datos, while permitirá almacenar los mismos dentro del arreglo, para finalmente, al momento de no encontrar más, retornar el mismo.



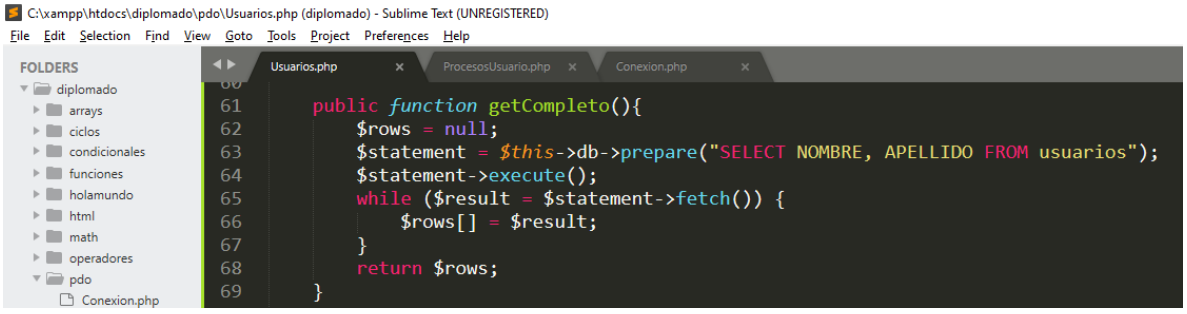


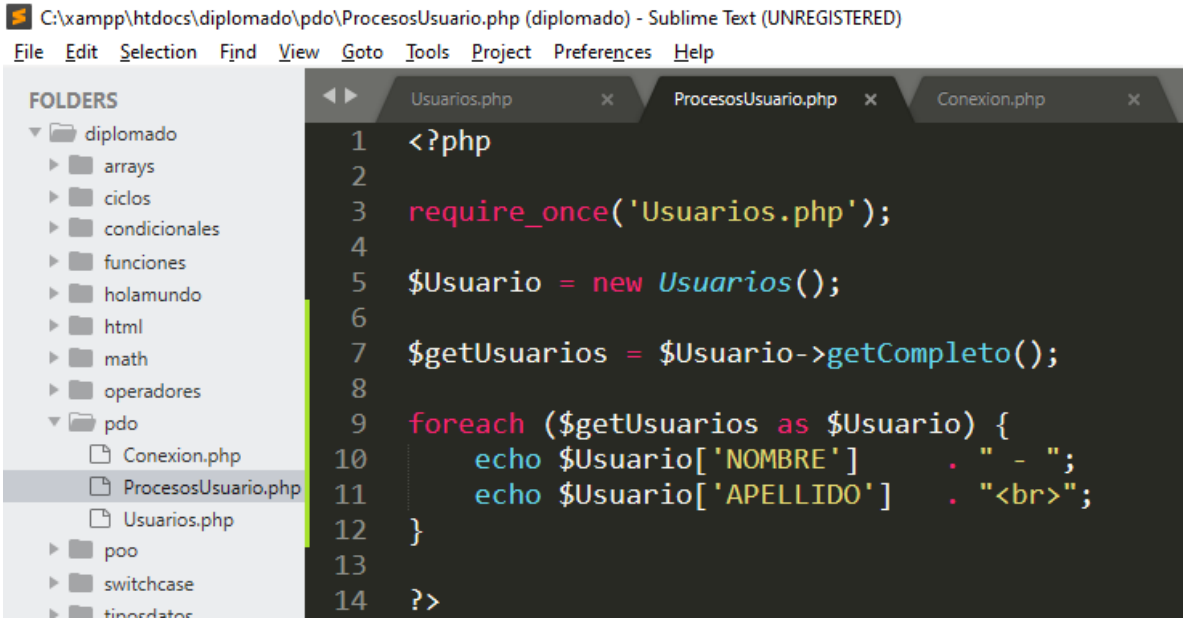
El proceso de impresión es bastante simple, es similar al visto en módulo pasado con relación a los arreglos, simplemente realizaremos la impresión por medio de un foreach y dando haciendo referencia a cada una de nuestras columnas de nuestra tabla.

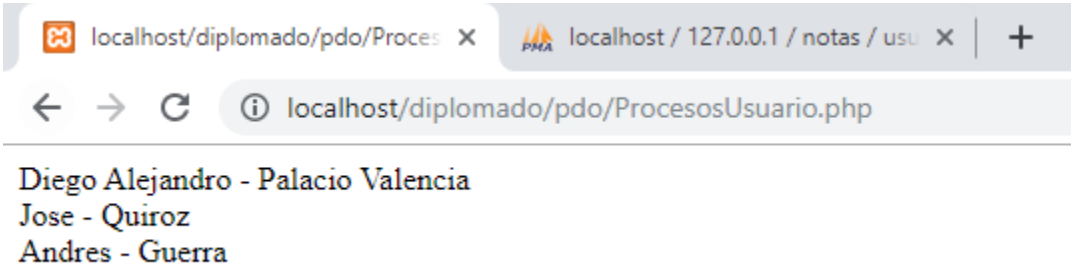
Sintaxis 2:



En la sintaxis 2, especificamos los campos que deseamos obtener por medio del **SELECT**, pueden ser todos, 3, 2 o 1, todo depende de la necesidad. El resto del proceso es el mismo, mostremos en este caso: **NOMBRE** y **APELLIDO**.




C:\xampp\htdocs\diplomado\pdo\ProcesosUsuario.php (diplomado) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
diplomado
arrays
ciclos
condicionales
funciones
holamundo
html
math
operadores
pdo
Conexion.php
ProcesosUsuario.php
Usuarios.php
poo
switchcase
Usuarios.php
ProcesosUsuario.php
Conexion.php
1 <?php
2
3 require_once('Usuarios.php');
4
5 $Usuario = new Usuarios();
6
7 $getUsuarios = $Usuario->getCompleto();
8
9 foreach ($getUsuarios as $Usuario) {
10 echo $Usuario['NOMBRE'] . " - ";
11 echo $Usuario['APELLIDO'] . "<br>";
12 }
13
14 ?>

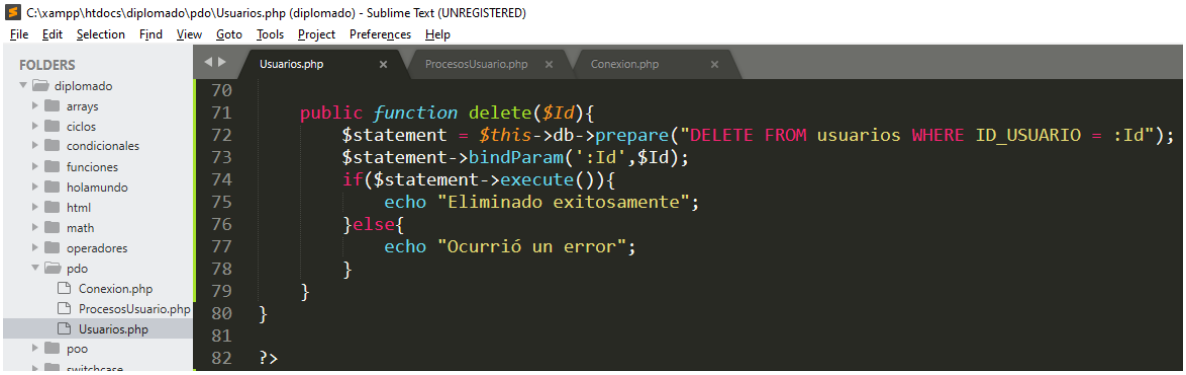

localhost/diplomado/pdo/Proces
localhost / 127.0.0.1 / notas / usu
localhost/diplomado/pdo/ProcesosUsuario.php
Diego Alejandro - Palacio Valencia
Jose - Quiroz
Andres - Guerra

Eliminar

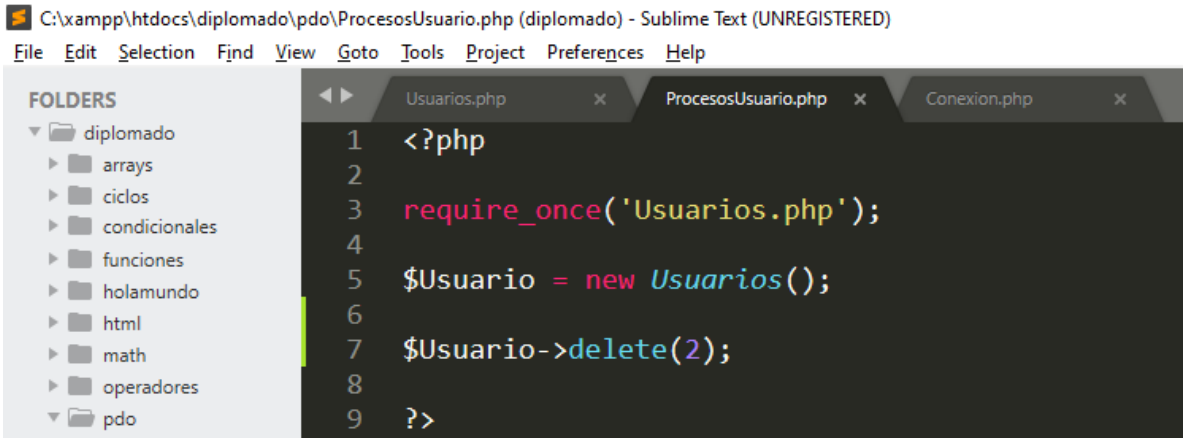
La función eliminar, permite eliminar los datos de una tabla. Consta de un método que realizará uso de las clases de PDO para operar sobre las tablas de nuestra base de datos. Sintaxis:

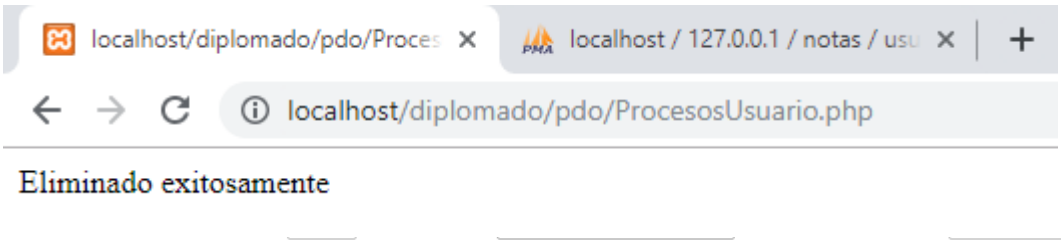
DELETE FROM nombre_tabla WHERE *columna_condicion* = *:value_condicion*

Veamos el código:



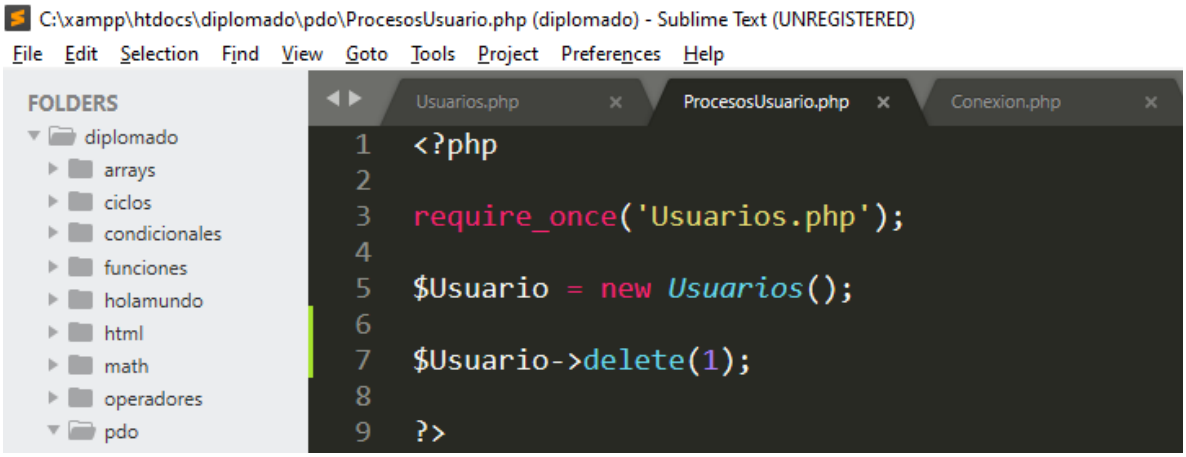
Llegados a este punto, es fácil entender el funcionamiento de la sentencia, la palabra reservada DELETE indica a la sentencia que se realizará una eliminación de información, simplemente se debe especificar el nombre de la tabla y la condicional para la eliminación. Veamos a ver:

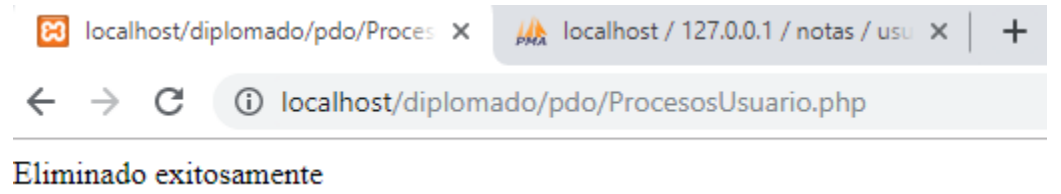




+ Opciones

	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
Editar Copiar Borrar	1	Diego Alejandro	Palacio Valencia	dp11ego	abc9292	Docente	Activo
Editar Copiar Borrar	3	Andres	Guerra	aagg46	123	Administrador	Inactivo





+ Opciones

	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
Editar Copiar Borrar	3	Andres	Guerra	aagg46	123	Administrador	Inactivo

TEMA 3
DESARROLLO DEL PROYECTO - ESTRUCTURA



Este apartado del diplomado se encuentra en vídeo, puedes verlo y entender un poco más el proceso. disponible en: https://www.youtube.com/watch?v=P7NQRut0X_w&feature=youtu.be.

Contando ya con nuestra base de datos implementada y con los conceptos básicos y funcionales de la extensión PDO podremos pasar a desarrollar nuestro proyecto.

La estructura que propongo para el desarrollo del mismo consiste en dividir el proyecto por MODULOS, es decir, cada posible entidad de nuestro proyecto se puede convertir en un MODULO, veamos:

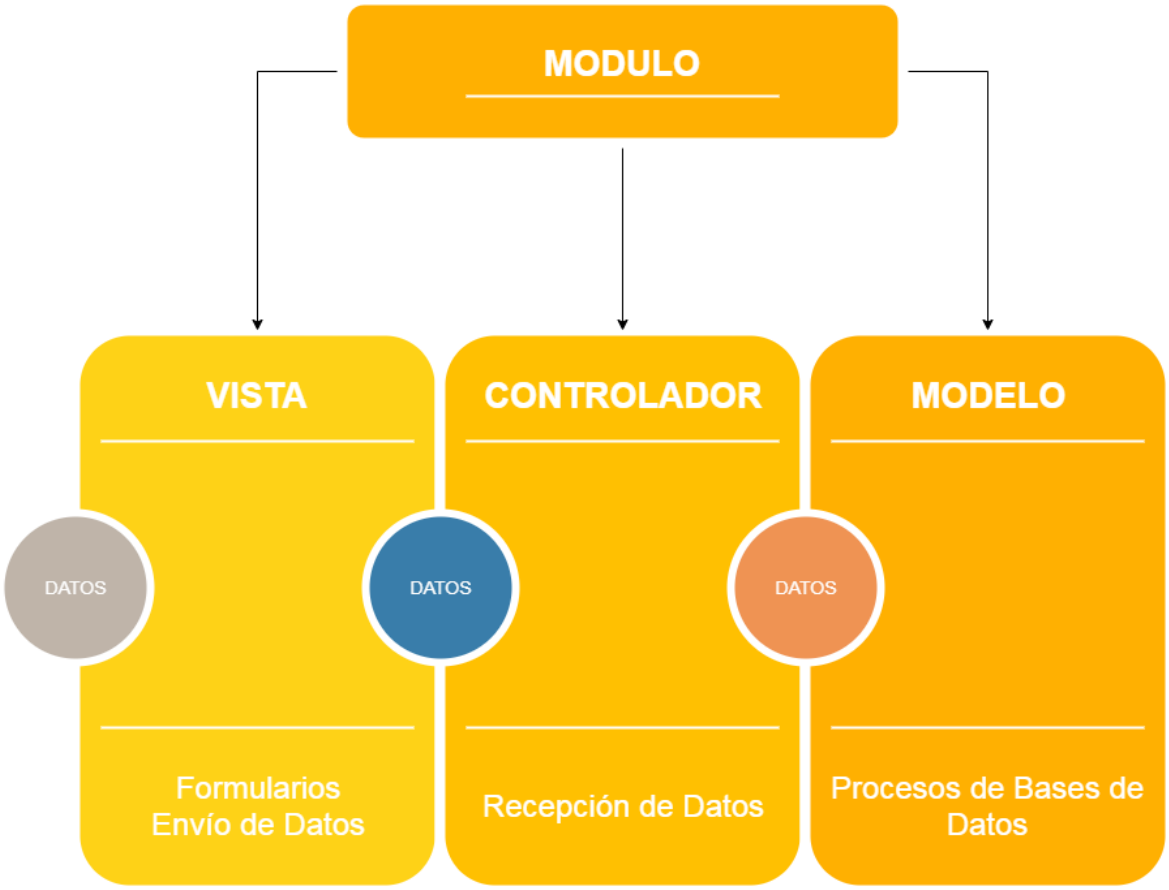
- Administradores
- Docentes
- Estudiantes
- Materias
- Usuarios

Para nuestro caso pueden ser estos, pero en caso de que tuviésemos más características y funcionalidades, estas se podrían ver envueltas completamente en un MODULO.

Nuestros módulos van a implementar la arquitectura de desarrollo de software MODELO – VISTA- CONTROLADOR, y para satisfacer esta arquitectura, nuestros MODULOS deben contar con una estructura organizada, constate y clara.

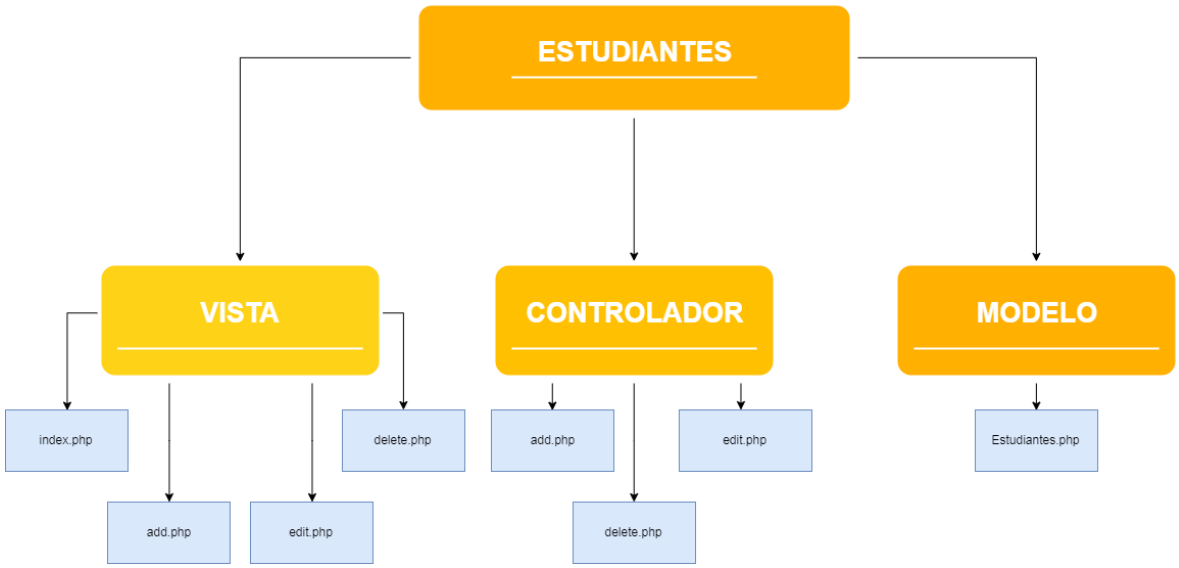
La arquitectura MVC se divide en tres bloques fundamentales:

- Modelo
- Vista
- Controlador

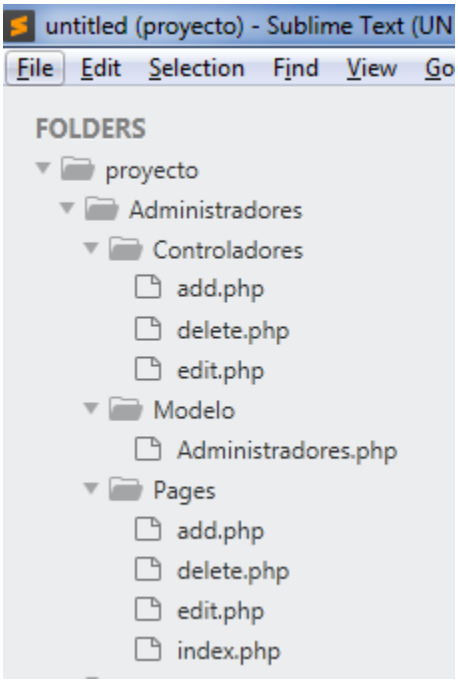


Esta es la forma general de ver un módulo basado en modelo vista controlador, todo el código se ve repartido en 3 bloques fundamentales con tareas y procesos completamente diferentes.

Esta es una forma muy general, para no entrar en mucho detalle para posteriormente detallar aún más esta arquitectura veamos un poco más general como opera.



Es decir, según este esquema, esta es la correcta forma de definir nuestras funcionalidades del proyecto, veamos ya un ejemplo más aterrizado.



Dentro de nuestro proyecto, para el ejemplo con el caso del MODULO de administradores, realizamos las respectivas divisiones:

- Controladores
- Modelo
- Pages

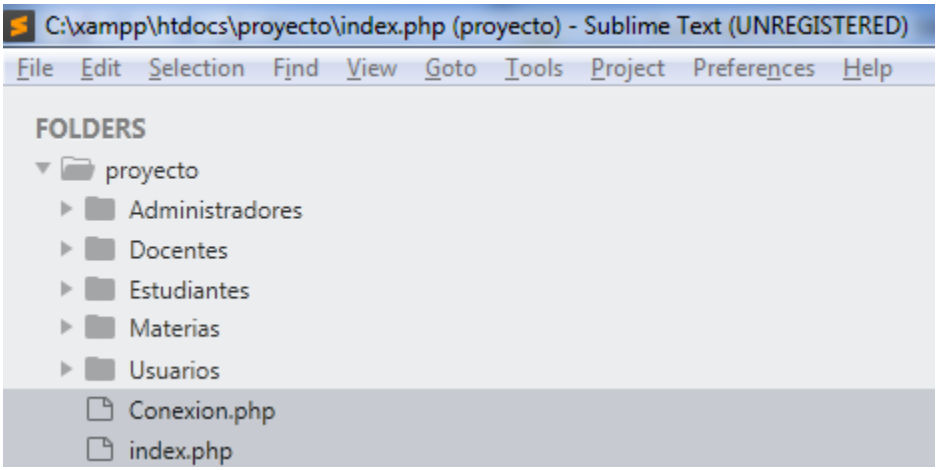
Y dentro de cada una de estas divisiones, se encuentra el código fuente y los archivos respectivos.

Así deben estar cada uno de estos para contener finalmente la estructura de nuestro proyecto.

Adicional a todo esto, hacen falta dos archivos importantes:

- Conexión.php
- Index.php

Que recordemos que nuestra clase Conexion permitirá heredar con todos los modelos la conexión para hacer uso de la misma y el archivo index, determinará la página inicial (Inicio de sesión) de nuestro sistema.



Vea completo el esquema de nuestro proyecto.



Con esta estructura, contaremos finalmente con el model/esqueleto/esquema de nuestro proyecto. A continuación diseñaremos los formularios y páginas.

TEMA 4
DESARROLLO DEL PROYECTO - FORMULARIOS



Este apartado del diplomado se encuentra en vídeo, puedes verlo y entender un poco más el proceso. disponible en: https://www.youtube.com/watch?v=P7NQRut0X_w&feature=youtu.be.

TEMA 5
DESARROLLO DEL PROYECTO – MVC



Este apartado del diplomado se encuentra en vídeo, puedes verlo y entender un poco más el proceso. disponible en: https://www.youtube.com/watch?v=tIfHz_RxiKE

TEMA 6
DESARROLLO DEL PROYECTO – FUNCIONALIDADES - 1



Este apartado del diplomado se encuentra en vídeo, puedes verlo y entender un poco más el proceso. disponible en: https://www.youtube.com/watch?v=qaQdne7uVYU

Recursos disponibles para el aprendizaje



Para desarrollar las habilidades y destrezas necesarias en cada competencia, es muy importante que tengas acceso a los recursos didácticos adecuados.

Entonces, si necesitas reforzar esta información, te sugerimos revisar nuevamente los **Vídeos de Apoyos**, disponibles en el campus virtual, indicados en las lecturas anteriores. Además, recuerda que puede consultar las **Fuentes Documentales** que aparecen en esta guía, particularemnte, en el apartado de Referencias Bibliográficas.

ASPECTOS CLAVES

- PDO es una extensión de PHP para administrar bases de datos.
- Las cuatro funciones básica para operar bases de datos son INSERT, SELECT, UPDATE y DELETE.
- PhpMyAdmin es la interfaz provista por XAMPP para administrar bases de datos.
- WHERE es una clausula para implementar condicionales en nuestras sentencias.
- BindParam permite referenciar valores de nuestros parametros y nuestra sentencias preparadas.
- Recordar el uso de POST y GET. La diferencia está en la forma de envíar la información.
- Execute() determina la ejecución de una sentencia preparada.

¡Felicidades! 👍 Has concluido con la lectura de la Guía Didáctica N°5. Así que ya puedes realizar el Foro 5.

Esta guía fue elaborada para ser utilizada con fines didácticos como material de consulta de los participantes en el Diplomado Virtual en Programación en PHP del Politécnico de Colombia, especialmente, a los técnicos, tecnólogos y profesionales de carreras afines, estudiantes de todas las carreras, empíricos, y público en general con conocimientos básicos en informática que intentan entrar en el mundo de la programación, que se desempeñen o no en las áreas de TIC de cualquier tipo de organización y que deseen obtener las competencias y habilidades necesarias para conocer los fundamentos prácticos del lenguaje de programación PHP para la aplicación y desarrollo de algoritmos y aplicaciones, y solo podrá ser reproducida con esos fines. Por lo tanto, se agradece a los usuarios referirla en los escritos donde se utilice la información que aquí se presenta.

Derechos reservados - POLITÉCNICO DE COLOMBIA, 2019
Medellín, Colombia