



Diplomado virtual en  
**PROGRAMACIÓN EN PHP**  
*Guía didáctica 5: Desarrollo web II*



Formación Virtual

.....educación sin límites



## Competencia específica

Se espera que, con los temas abordados en la guía didáctica del módulo 5: Desarrollo web II, el estudiante logre la siguiente competencia específica:

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



## Contenidos temáticos

Los contenidos temáticos, para desarrollar en la guía didáctica del módulo 5: Desarrollo web II, son:

Bases de datos - MySQL (parte II)

PHP - PDO

Desarrollo del proyecto - estructura

Desarrollo del proyecto - formularios

Desarrollo del proyecto – MVC

Desarrollo del proyecto – funcionalidades (I)

Desarrollo del proyecto – funcionalidades (II)

Desarrollo del proyecto – funcionalidades (III)

### **Ilustración 1: caracterización de la guía didáctica.**

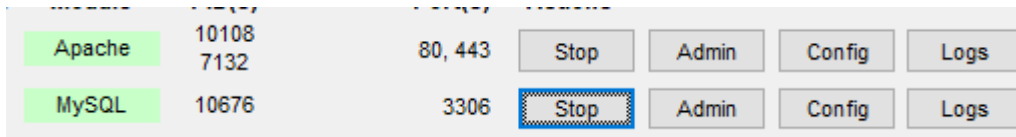
Fuente: autor<sup>1</sup>.

---

<sup>1</sup> Todas las ilustraciones y tablas de esta guía son autoría propia del docente y tienen como función mostrar la aplicación y práctica del contenido que se desarrolla, por ello solo se enumerarán.

## Tema 1: Bases de Datos - MySQL (parte II)

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:



Apache	10108 7132	80, 443	Stop	Admin	Config	Logs
MySQL	10676	3306	Stop	Admin	Config	Logs

### Ilustración 2.

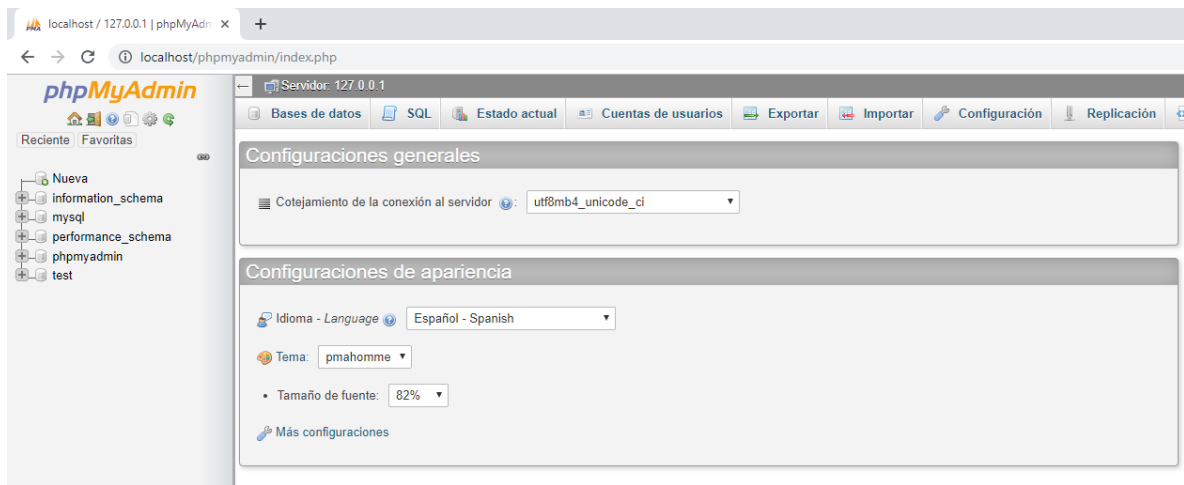
phpMyAdmin es una herramienta gratuita que permite de una manera muy completa acceder a todas las funciones de la base de datos MySQL a través de PHP, mediante una interfaz web muy intuitiva.

### ¿Qué ofrece phpMyAdmin?

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

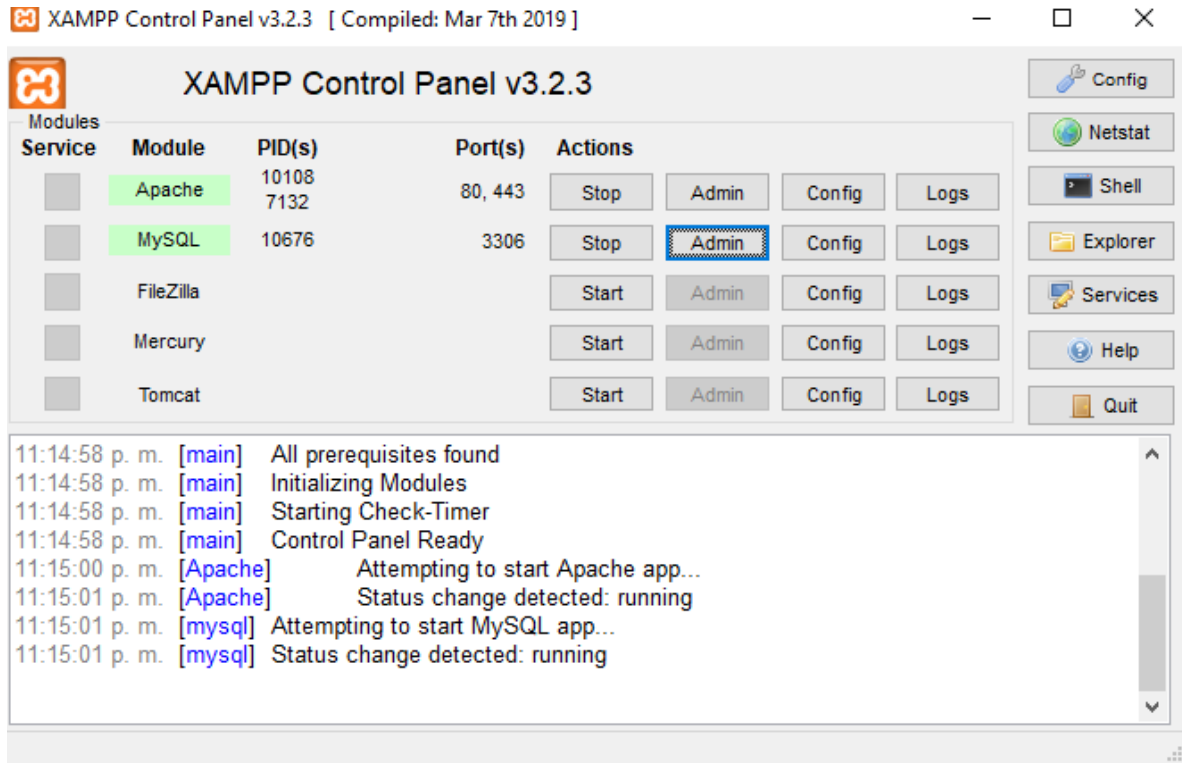
- Esta aplicación nos permitirá realizar las operaciones básicas en base de datos MySQL, como son: crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar sentencias SQL, administrar claves de campos, administrar privilegios y exportar datos en varios 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 test de forma rápida y sencilla.



**Ilustración 3.**

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



**Ilustración 4.**

También es válido ir 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:

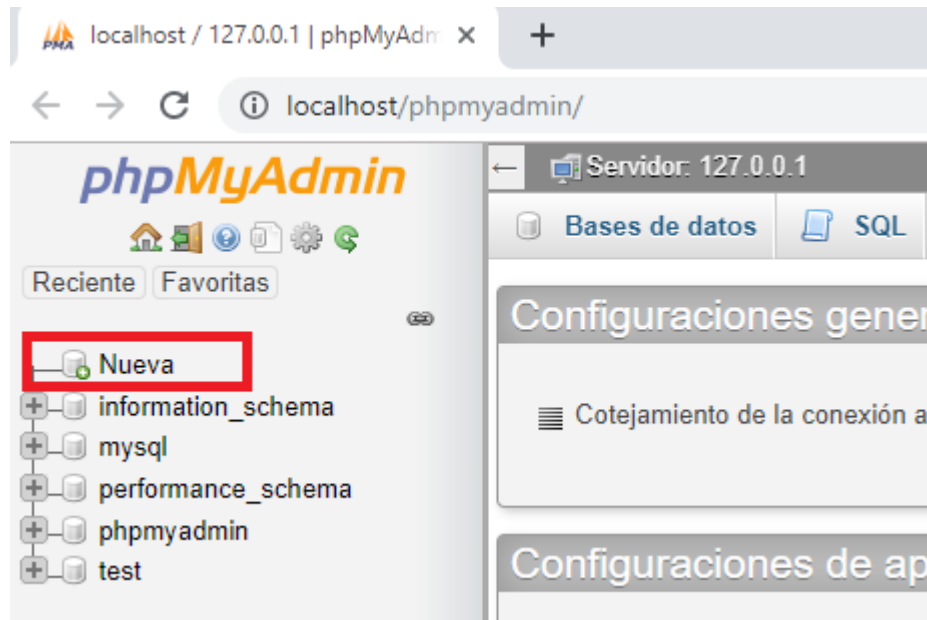
- Creación de bases de datos.
- Creación de tablas.
- Especificación de campos de tabla.
- Tipos de datos.
- Claves primarias.
- Inserción de datos.
- Consulta de datos.
- Actualización de datos.
- Eliminación de datos.
- Exportación de datos y tablas.



- Copias de seguridad.
- Modelo Entidad Relación.

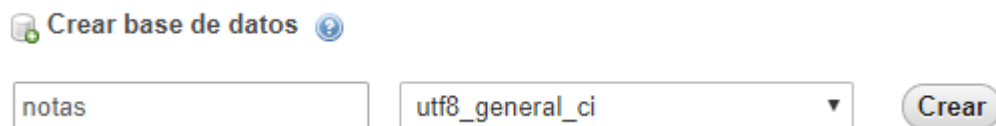
Para crear nuestra primera base de datos, que será la utilizada en el proyecto final, el proceso que debemos realizar es el siguiente:

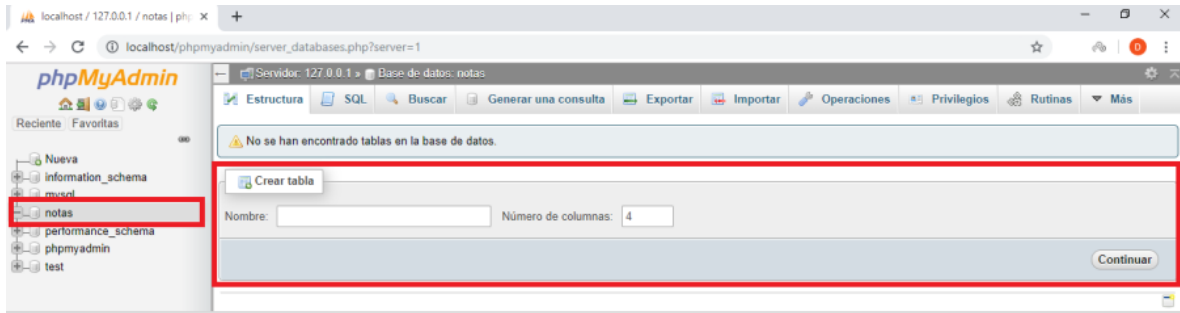
- Abrir phpMyAdmin.
- En el apartado superior izquierdo se encuentran las bases de datos establecidas por defecto de MySQL y las que el desarrollador vaya ocupando en su labor. Ir a la opción de «Nueva».



**Ilustración 5.**

- 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:

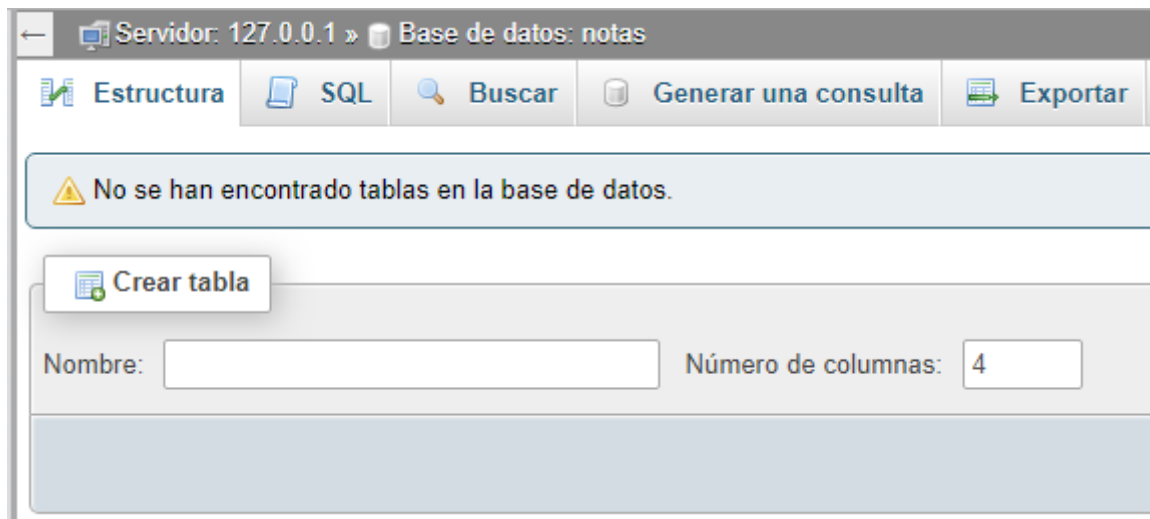




**Ilustraciones 6 y 7.**

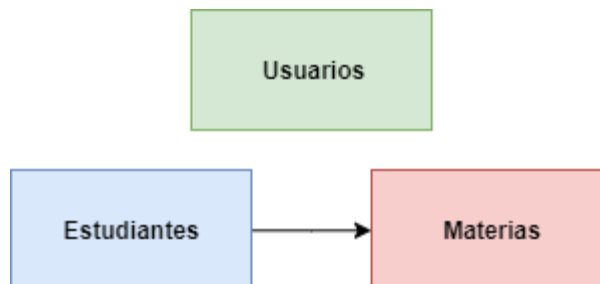
En este momento empezaremos el diseño de nuestro proyecto a partir de los temas previstos en los temas de este módulo, iniciando por nuestra base de datos.

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



**Ilustración 8.**

Con base en nuestra base de datos, y más específicamente nuestro proyecto, contaremos con cuatro tablas en la siguiente estructura:



**Ilustración 9.**

**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

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

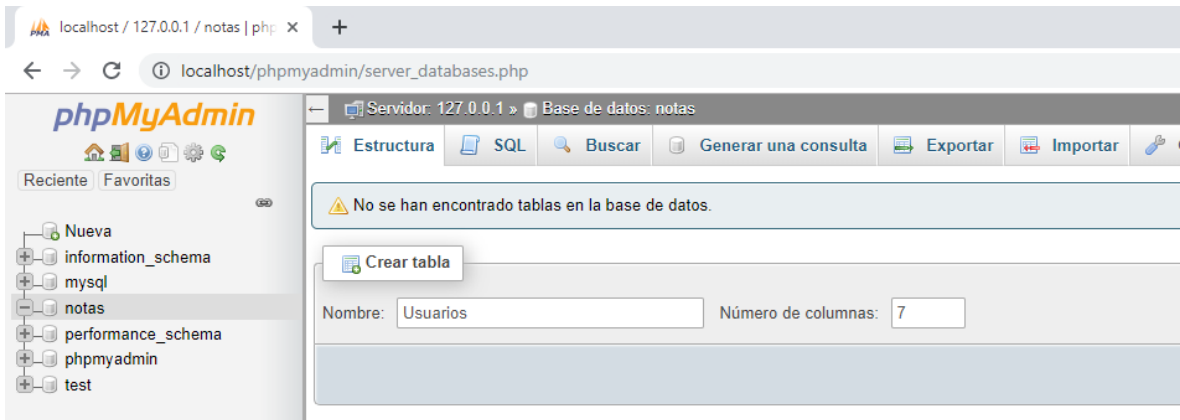
### Ilustración 10.

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

- El nombre de esta será «Usuarios», el nombre es claro y diciente de la información que contendrá.
- Tiene siete 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 la 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 a 1.



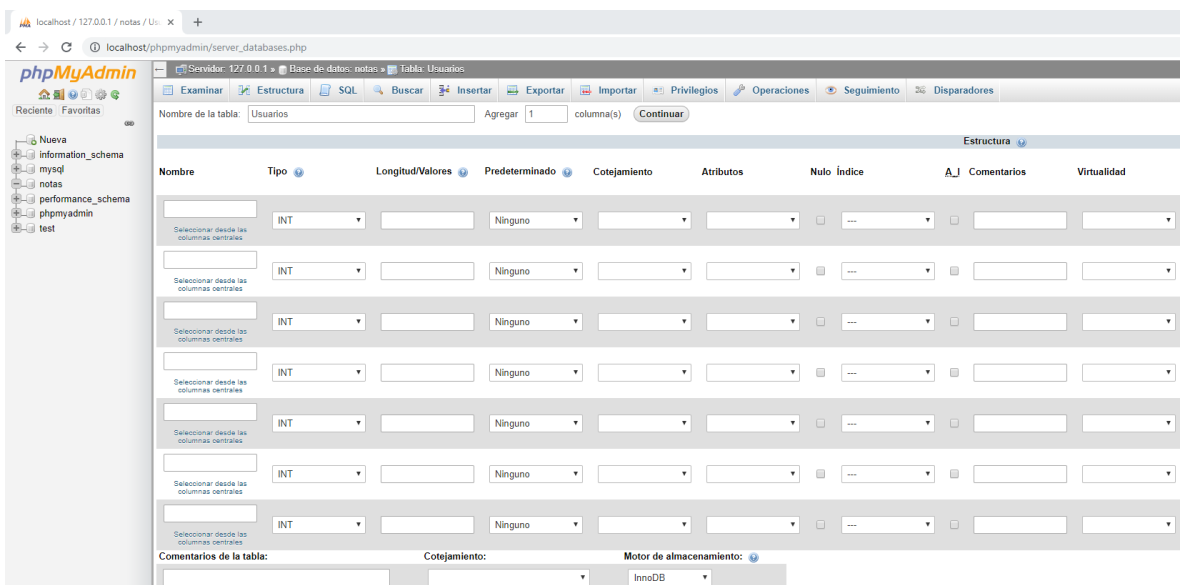
Para realizar la creación de la tabla «Usuarios» dentro de la base de datos «notas», regresemos al apartado de phpMyAdmin donde quedamos:



**Ilustración 11.**

- Debemos 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:

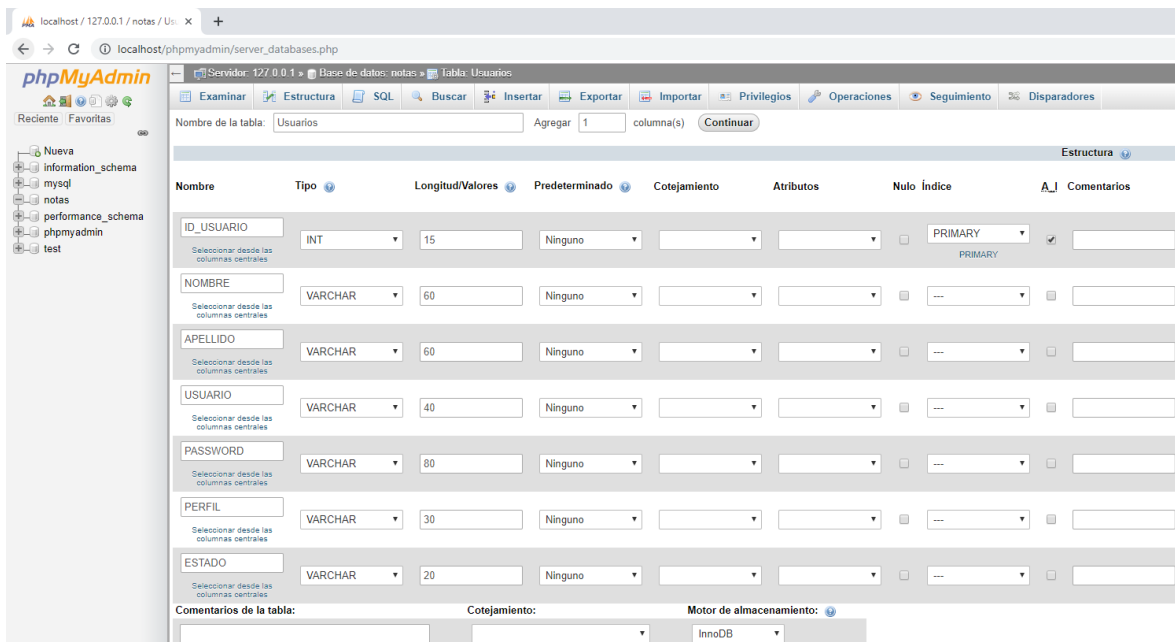


**Ilustración 12.**

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	-	-

Tabla 1.



The screenshot shows the phpMyAdmin interface for a MySQL database. The 'Estructura' (Structure) tab is selected for the 'Usuarios' table. The table structure is as follows:

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento	Atributos	Nulo	Índice	Comentarios
ID_USUARIO	INT	15	Ninguno			<input type="checkbox"/>	PRIMARY	
NOMBRE	VARCHAR	60	Ninguno			<input type="checkbox"/>		
APELLIDO	VARCHAR	60	Ninguno			<input type="checkbox"/>		
USUARIO	VARCHAR	40	Ninguno			<input type="checkbox"/>		
PASSWORD	VARCHAR	80	Ninguno			<input type="checkbox"/>		
PERFIL	VARCHAR	30	Ninguno			<input type="checkbox"/>		
ESTADO	VARCHAR	20	Ninguno			<input type="checkbox"/>		

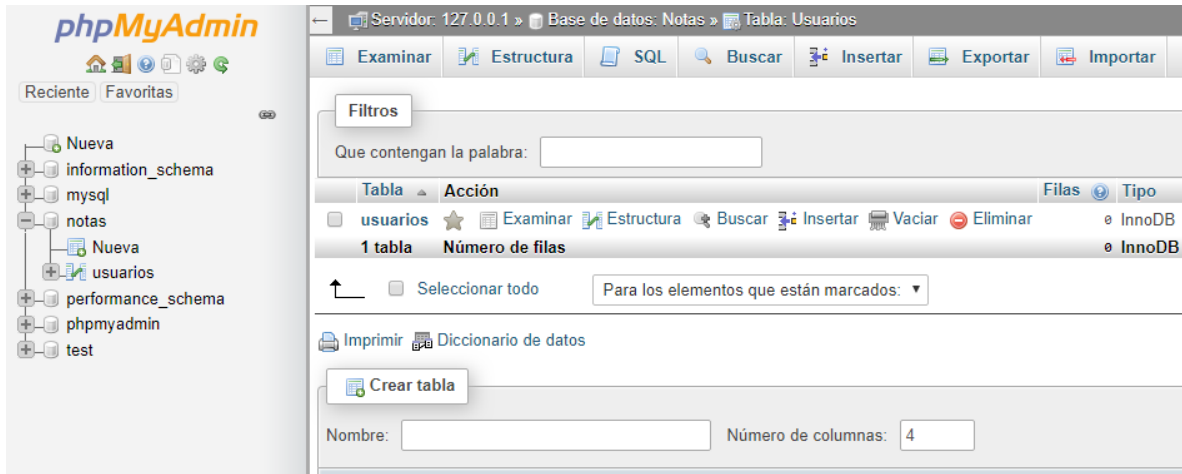
At the bottom, the 'Motor de almacenamiento' (Storage Engine) is set to 'InnoDB'.

Ilustración 13.

**Int:** permite almacenar únicamente valores numéricos.

**Varchar:** permite almacenar cadenas de texto.

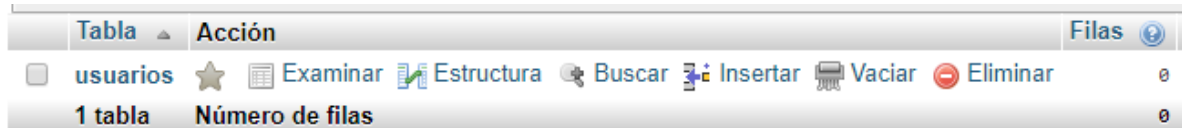
El resultado gráfico, luego de guardar los respectivos cambios, será el siguiente:



**Ilustración 14.**

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

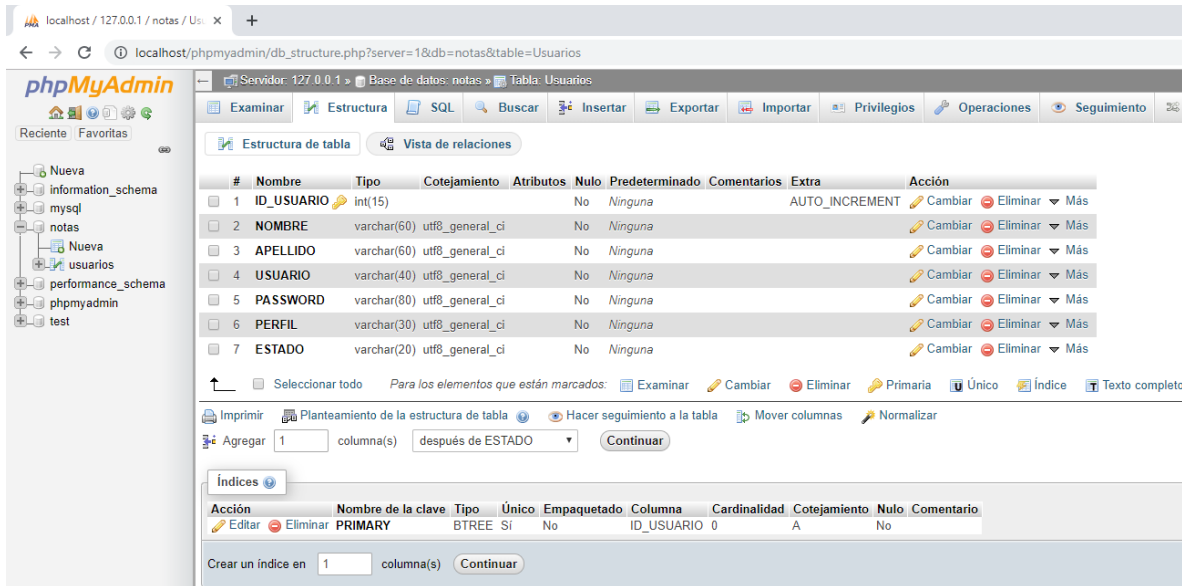
Esta tabla ofrece ya ciertas funciones para operar sobre la misma, como lo son:



**Ilustración 15.**

- 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:



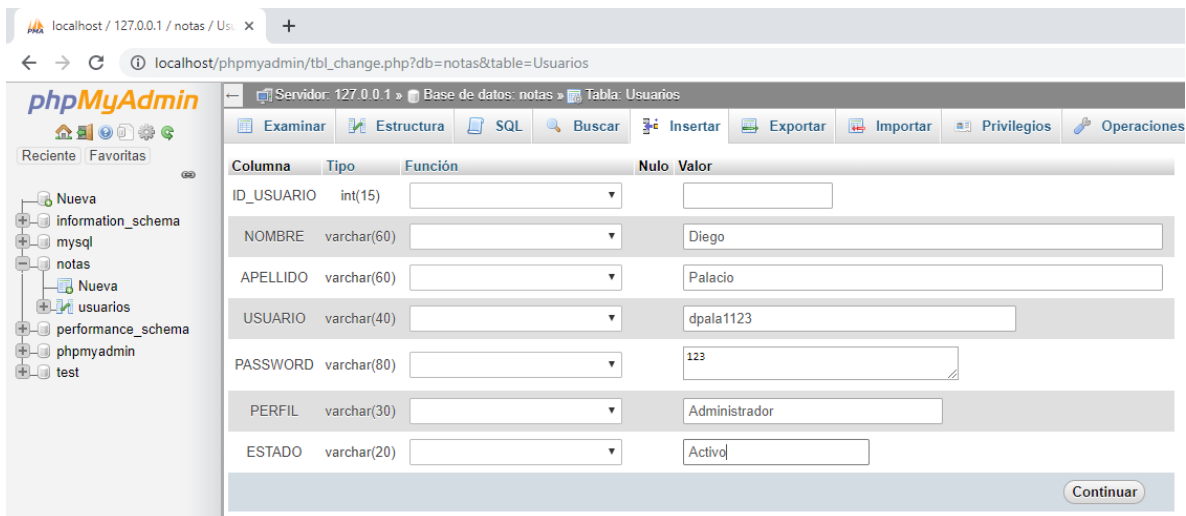
**Ilustración 16.**

**Insertar datos.** La primera función, y la principal en manejo de tablas y de datos, es la inserción de estos, podemos insertar datos de tres 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».



### Ilustración 17.

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:

✓ 1 fila insertada.  
La Id de la fila insertada es: 1

### Ilustración 18.

```
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.

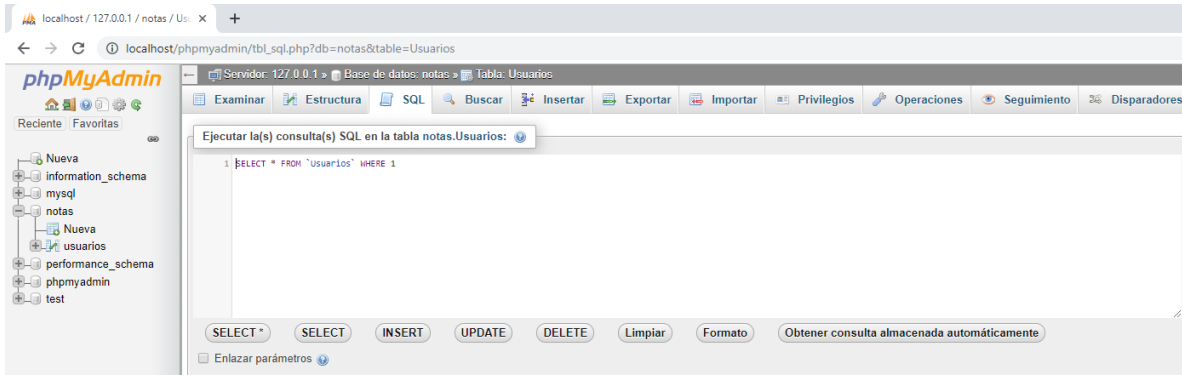


The screenshot shows the phpMyAdmin interface for a database named 'notas'. The 'Usuarios' table is selected, and the 'Examinar' (Browse) tab is active. A green message bar indicates 'Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0000 segundos.)'. Below this, a table displays the data:

ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
1	Diego	Palacio	dpala1123	123	Administrador	Activo

### Ilustración 19.

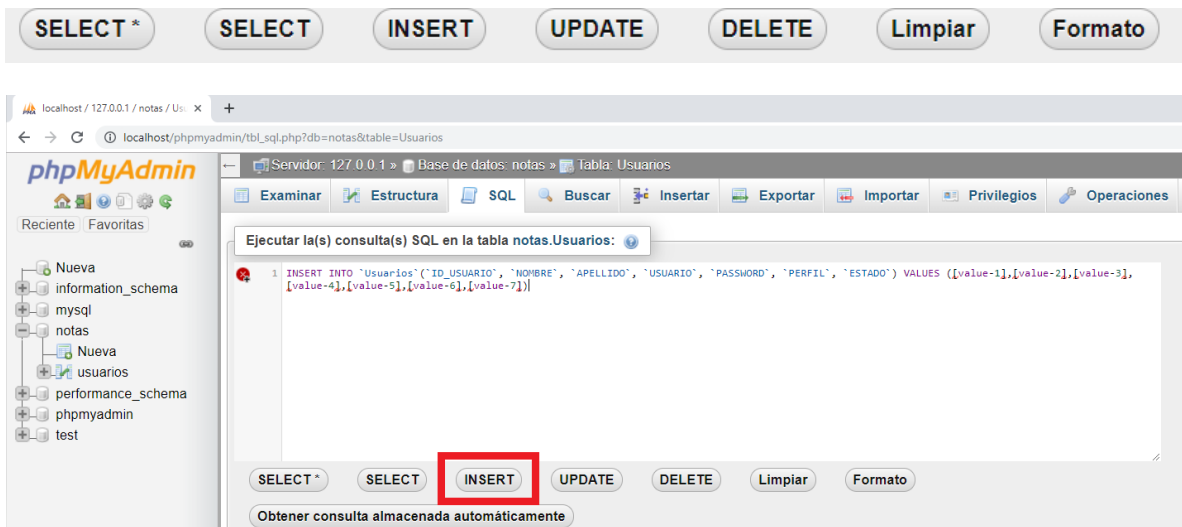
**Consola de código.** La consola de código ofrece una interfaz para realizar directamente consultas a una base de datos y más directamente sobre las tablas.



**Ilustración 20.**

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 interfaz, para este caso, usaremos «Insert».

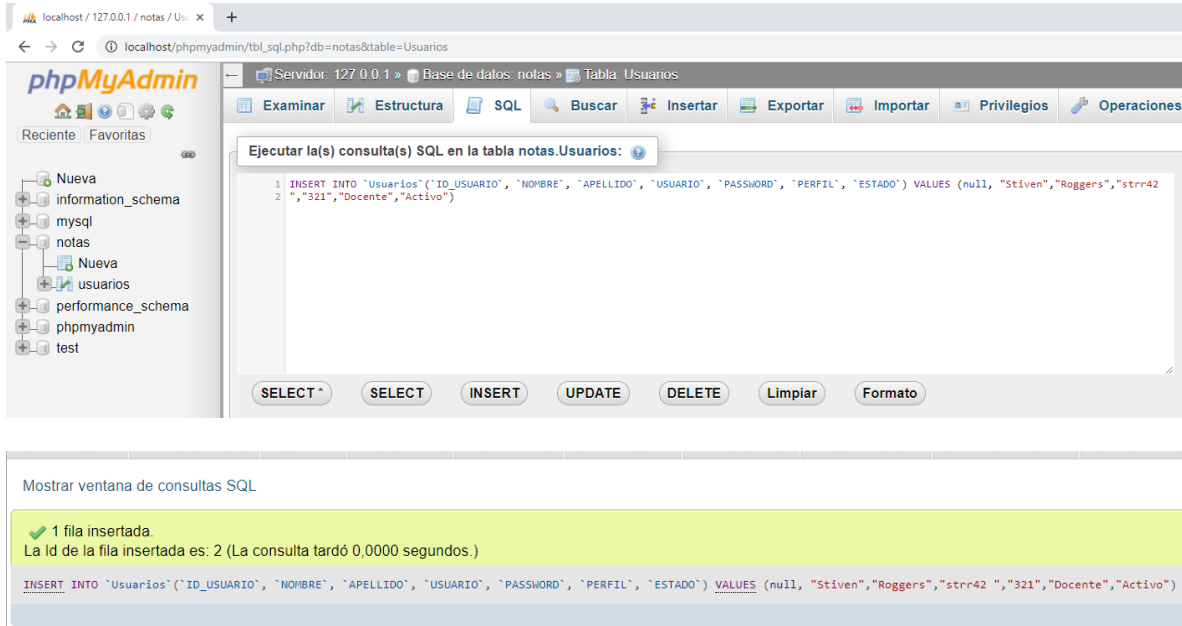


**Ilustraciones 21 y 22.**

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","str42","321","Docente","Activo")
```





localhost / 127.0.0.1 / notas / Us: X

localhost/phpmyadmin/tbl\_sql.php?db=notas&table=Usuarios

Servidor: 127.0.0.1 » Base de datos: notas » Tabla: Usuarios

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones

Ejecutar la(s) consulta(s) SQL en la tabla notas.Usuarios:

```
1 INSERT INTO `Usuarios` (`ID_USUARIO`, `NOMBRE`, `APELLIDO`, `USUARIO`, `PASSWORD`, `PERFIL`, `ESTADO`) VALUES (null, "Stiven", "Roggers", "str42", "321", "Docente", "Activo")
2
```

SELECT \* SELECT INSERT UPDATE DELETE Limpiar Formato

Mostrar ventana de consultas SQL

✓ 1 fila insertada.  
La Id de la fila insertada es: 2 (La consulta tardó 0,0000 segundos.)

INSERT INTO `Usuarios` (`ID\_USUARIO`, `NOMBRE`, `APELLIDO`, `USUARIO`, `PASSWORD`, `PERFIL`, `ESTADO`) VALUES (null, "Stiven", "Roggers", "str42 ", "321", "Docente", "Activo")

### Ilustraciones 23 y 24.

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



localhost / 127.0.0.1 / notas / Us: X

localhost/phpmyadmin/sql.php?db=notas&table=Usuarios&pos=0

Servidor: 127.0.0.1 » Base de datos: notas » Tabla: Usuarios

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones

✓ Mostrando filas 0 - 1 (total de 2, La consulta tardó 0,0000 segundos.)

SELECT \* FROM `Usuarios`

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

+ Opciones

	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Diego	Palacio	dpala1123	123	Administrador	Activo
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Stiven	Roggers	str42	321	Docente	Activo

Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

### Ilustración 25.

El resultado ahora serán dos datos insertados de dos formas diferentes, igual de funcionales, igual de prácticos. Recomendando realizar estos 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 evaluarán en el sistema. La información que contendrá nuestra tabla será:

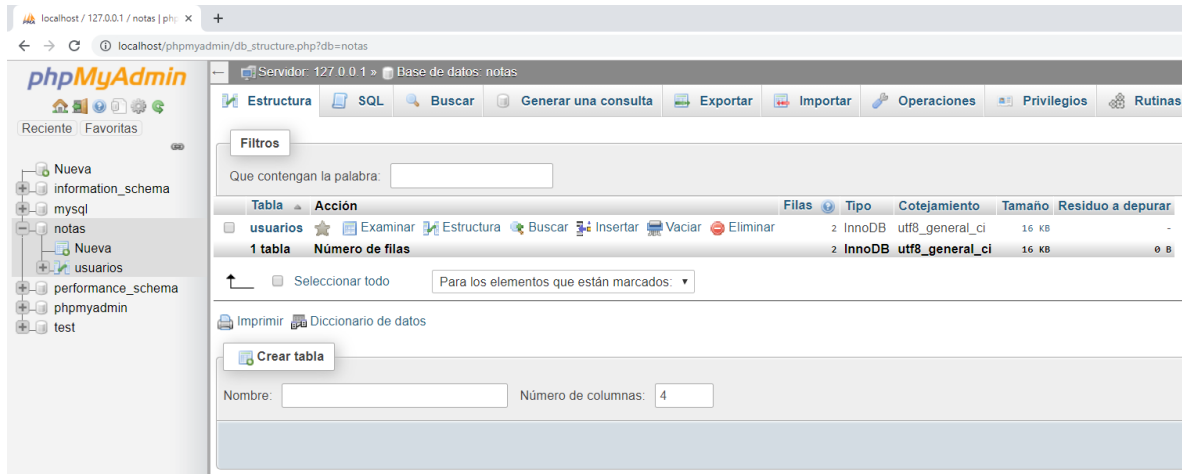
Materias	
ID_MATERIA	MATERIA
1	Matematicas
2	Ciencias
3	Fisica
4	Ingles

#### **Ilustración 26.**

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


- El nombre de esta será «Materias», el nombre es claro y diciente de la información que contendrá.
- Tiene dos 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 a 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:



**Ilustración 27.**

Simplemente, de nuevo, asignamos nombre y número de columnas:


**Crear tabla**

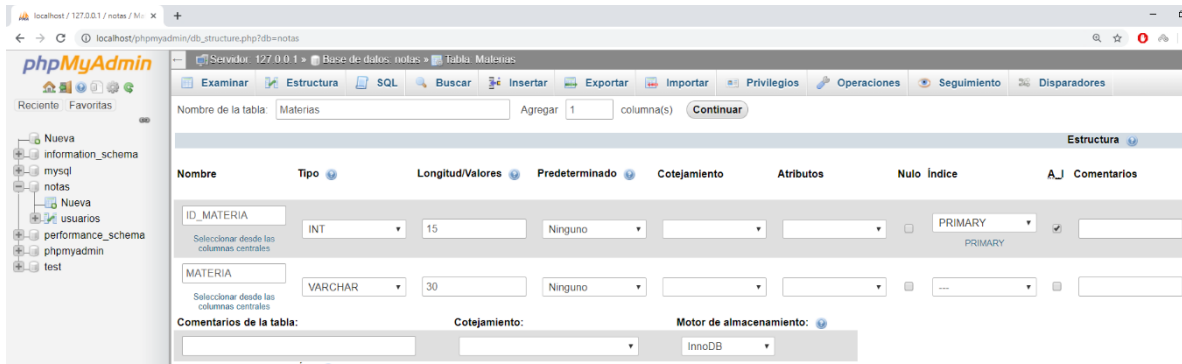
Nombre: 
Número de columnas:

**Ilustración 28.**

Definiendo las características de nuestra tabla, serían las siguientes:

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

**Tabla 2.**



**Ilustración 29.**

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



**Ilustración 30.**

Realiza la inserción de las cuatro respectivas materias de prueba adjuntadas en la guía: matemáticas, ciencias, física e inglés, con sus respectivos ID.

## 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:

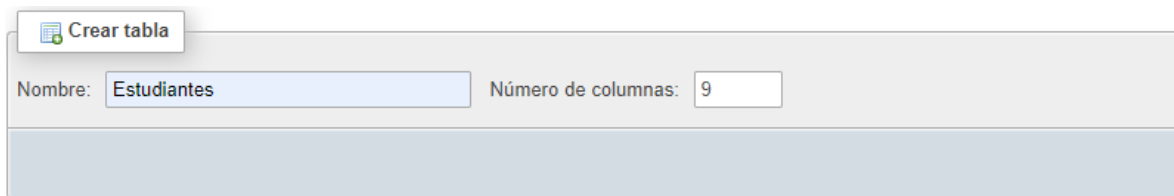
Estudiantes								
ID_ESTUDIANTE	NOMBRE	APELLIDO	DOCUMENTO	CORREO	MATERIA	DOCENTE	PROMEDIO	FECHA_REGISTRO
1	Estefania	Lopez	1004321424	estefi@gmail.com	Ingles	Stiven	90	2019-10-23
2	Evelin	Gutierrez	4314216	evegu@gmail.com	Fisica	Diego	80	2019-08-02

**Ilustración 31.**

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

- El nombre de esta será «Estudiantes», el nombre es claro y diciente de la información que contendrá.
- Tiene nueve 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 Estefanía 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 a 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:



**Ilustración 32.**

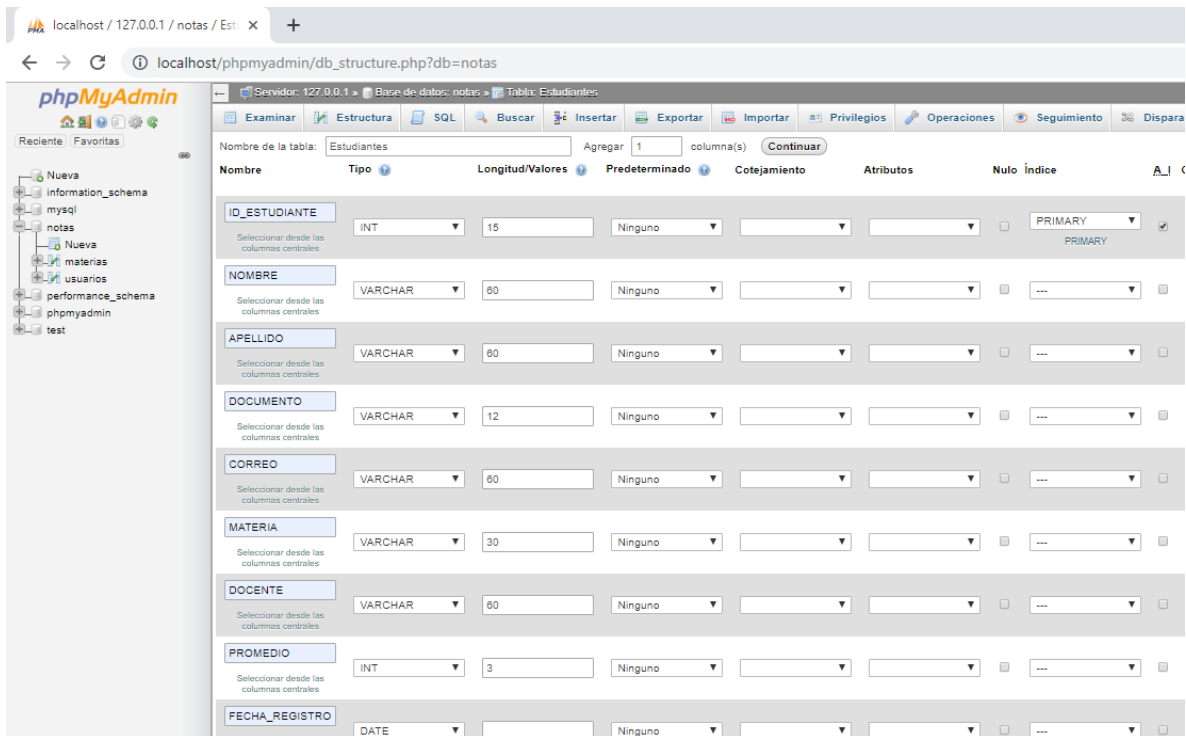
Definiendo las características de nuestra tabla, serían las siguientes:

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	-	-

<b>MATERIA</b>	VARCHAR	30	-	-
<b>DOCENTE</b>	VARCHAR	60	-	-
<b>PROMEDIO</b>	INT	3	-	-
<b>FECHA_REGISTRO</b>	DATE	-	-	-

**Tabla 3.**

**Date:** permite almacenar fechas en formato: YYYY-MM-DD.



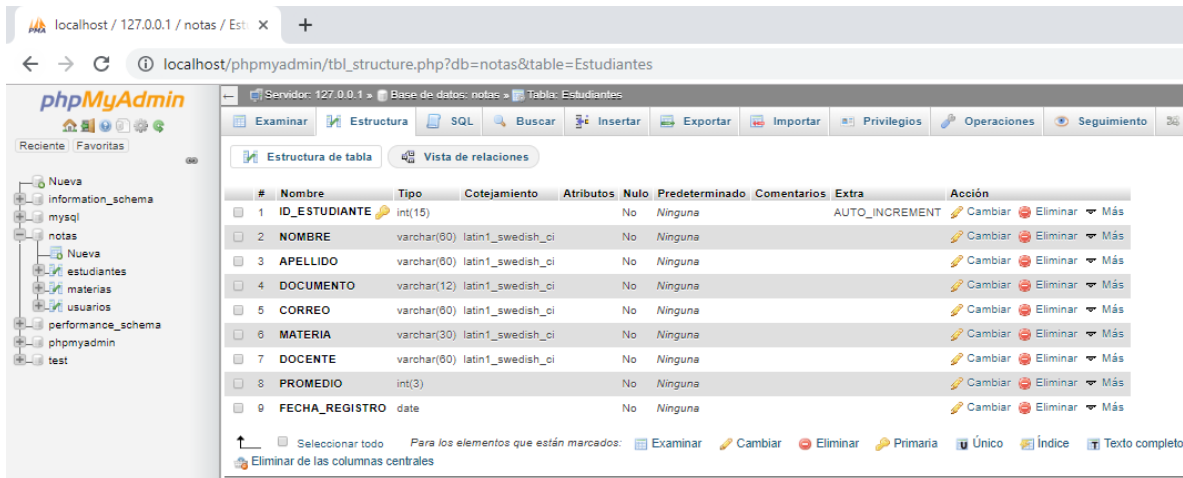
The screenshot shows the phpMyAdmin interface for a database named 'notas'. The table 'Estudiantes' is selected, and its structure is being edited. The table has the following columns:

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento	Atributos	Nulo	índice
ID_ESTUDIANTE	INT	15	Ninguno			<input type="checkbox"/>	PRIMARY
NOMBRE	VARCHAR	60	Ninguno			<input type="checkbox"/>	
APELLIDO	VARCHAR	60	Ninguno			<input type="checkbox"/>	
DOCUMENTO	VARCHAR	12	Ninguno			<input type="checkbox"/>	
CORREO	VARCHAR	60	Ninguno			<input type="checkbox"/>	
MATERIA	VARCHAR	30	Ninguno			<input type="checkbox"/>	
DOCENTE	VARCHAR	60	Ninguno			<input type="checkbox"/>	
PROMEDIO	INT	3	Ninguno			<input type="checkbox"/>	
FECHA_REGISTRO	DATE		Ninguno			<input type="checkbox"/>	

**Ilustración 33.**

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



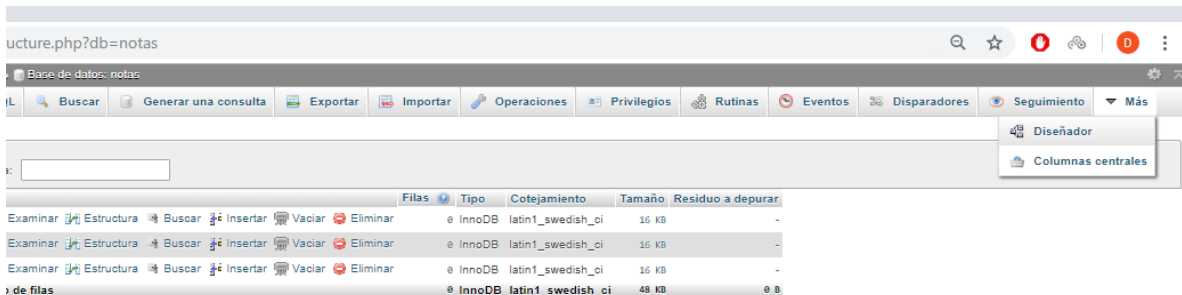


**Ilustración 34.**

Realiza la inserción de los dos respectivos estudiantes de prueba adjuntas en la guía: Estefanía y Evelin, con sus respectivos ID.

En este punto del proceso debemos contar ya con nuestra base de datos completa, las tres 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»:



**Ilustración 35.**

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.

<div> <div>notas usuarios</div> <div> <div>ID_USUARIO : int(15)</div> <div>NOMBRE : varchar(60)</div> <div>APELLIDO : varchar(60)</div> <div>USUARIO : varchar(40)</div> <div>PASSWORD : varchar(80)</div> <div>PERFIL : varchar(30)</div> <div>ESTADO : varchar(20)</div> </div> </div>	
<div> <div>notas estudiantes</div> <div> <div>ID_ESTUDIANTE : int(15)</div> <div>NOMBRE : varchar(60)</div> <div>APELLIDO : varchar(60)</div> <div>DOCUMENTO : varchar(12)</div> <div>CORREO : varchar(60)</div> <div>MATERIA : varchar(30)</div> <div>DOCENTE : varchar(60)</div> <div>PROMEDIO : int(3)</div> <div>FECHA_REGISTRO : date</div> </div> </div>	<div> <div>notas materias</div> <div> <div>ID_MATERIA : int(15)</div> <div>MATERIA : varchar(30)</div> </div> </div>

### Ilustración 36.

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:

```

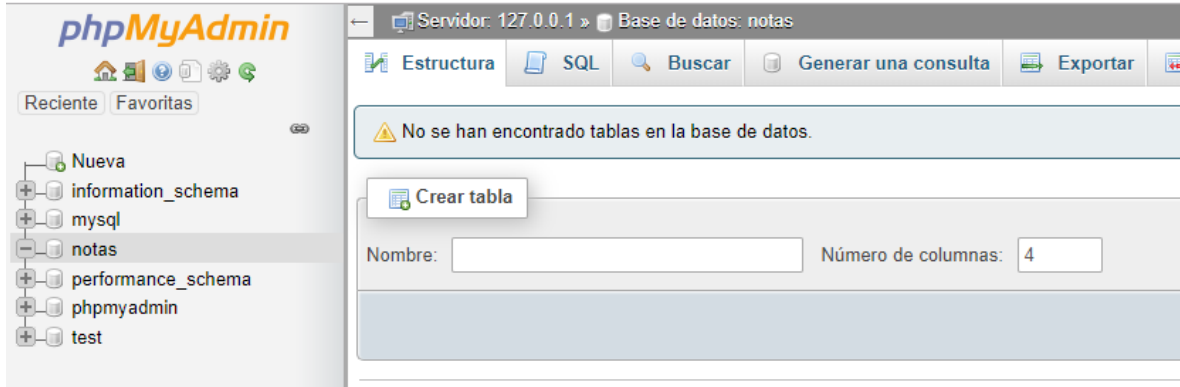
C:\Users\POLITECNICO\Dropbox\notas.sql (proyecto) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  proyecto
    Administradores
    Docentes
    Estudiantes
    Materias
    Usuarios
    Conexion.php
    index.php

notas.sql
1
2 CREATE TABLE usuarios
3 (
4     ID_USUARIO INT(15) NOT NULL AUTO_INCREMENT PRIMARY KEY,
5     NOMBRE VARCHAR(60) NOT NULL,
6     APELLIDO VARCHAR(60) NOT NULL,
7     USUARIO VARCHAR(40) NOT NULL,
8     PASSWORD VARCHAR(80) NOT NULL,
9     PERFIL VARCHAR(30) NOT NULL,
10    ESTADO VARCHAR(20) NOT NULL
11 );
12
13 CREATE TABLE materias
14 (
15     ID_MATERIA INT(15) NOT NULL AUTO_INCREMENT PRIMARY KEY,
16     MATERIA VARCHAR(30) NOT NULL
17 );
18
19 CREATE TABLE estudiantes
20 (
21     ID_ESTUDIANTE INT(15) NOT NULL AUTO_INCREMENT PRIMARY KEY,
22     NOMBRE VARCHAR(60) NOT NULL,
23     APELLIDO VARCHAR(60) NOT NULL,
24     DOCUMENTO VARCHAR(12) NOT NULL,
25     CORREO VARCHAR(60) NOT NULL,
26     MATERIA VARCHAR(30) NOT NULL,
27     DOCENTE VARCHAR(60) NOT NULL,
28     PROMEDIO INT(3) NOT NULL,
29     FECHA_REGISTRO DATE NOT NULL
30 );
  
```

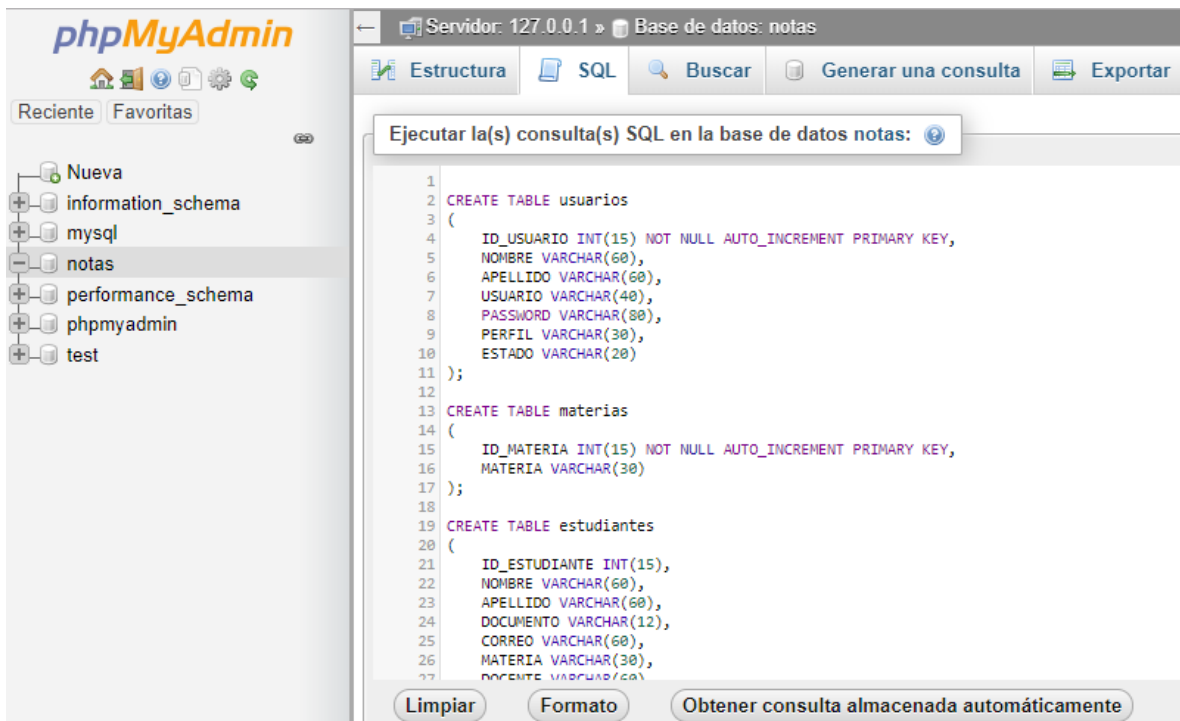
### Ilustración 37.

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



### Ilustración 38.

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



### Ilustración 39.

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

```

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0000 segundos.)
CREATE TABLE usuarios ( ID_USUARIO INT(15) NOT NULL AUTO_INCREMENT PRIMARY KEY, NOMBRE VARCHAR(60), APELLIDO VARCHAR(60), USUARIO VARCHAR(40), PASSWORD VARCHAR(80), PERFIL VARCHAR(30), ESTADO VARCHAR(20) )
[Editar en línea] [Editar] [ Crear código PHP ]

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0000 segundos.)
CREATE TABLE materias ( ID_MATERIA INT(15) NOT NULL AUTO_INCREMENT PRIMARY KEY, MATERIA VARCHAR(30) )
[Editar en línea] [Editar] [ Crear código PHP ]

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0000 segundos.)
CREATE TABLE estudiantes ( ID_ESTUDIANTE INT(15), NOMBRE VARCHAR(60), APELLIDO VARCHAR(60), DOCUMENTO VARCHAR(12), CORREO VARCHAR(60), MATERIA VARCHAR(30), DOCENTE VARCHAR(60), PROMEDIO INT(3), FECHA_REGISTRO DATE )
[Editar en línea] [Editar] [ Crear código PHP ]

```

#### Ilustración 40.

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

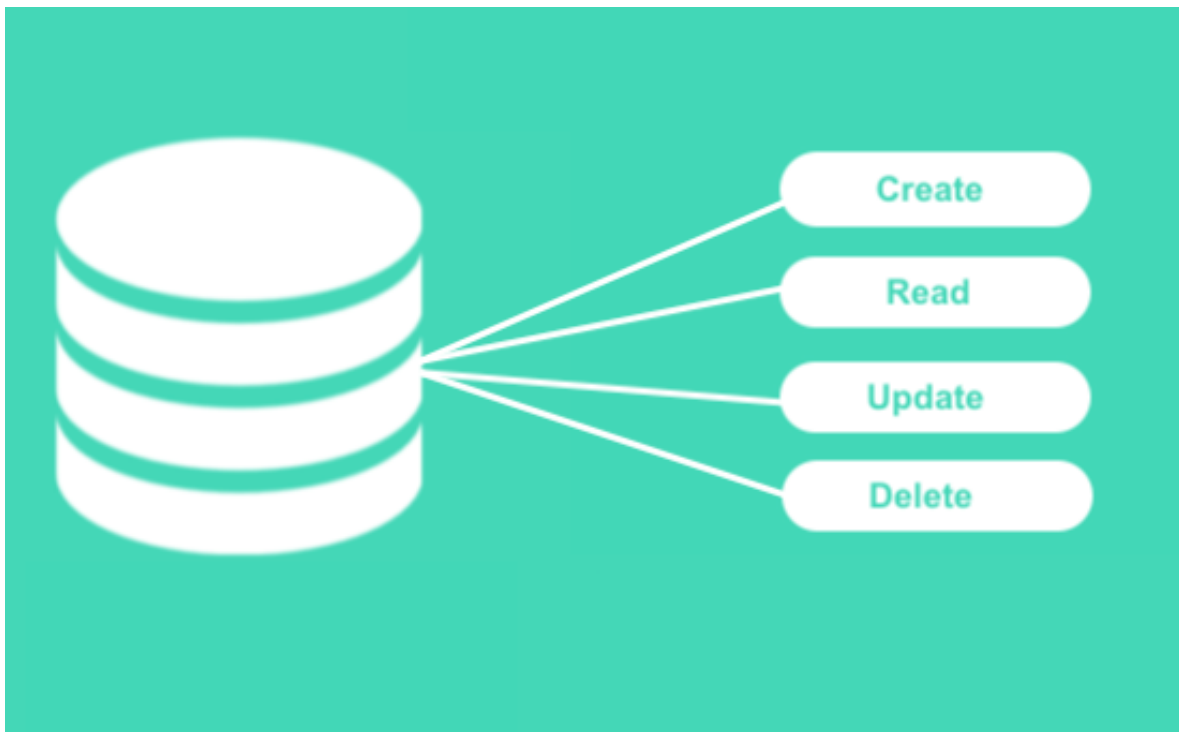
Filtros							
Que contengan la palabra: <input type="text"/>							
Tabla	Acción						Filas
<input type="checkbox"/> <b>estudiantes</b>	★	Examinar	✓ Estructura	🔍 Buscar	➕ Insertar	🗑 Vaciá	🚫 Eliminar
<input type="checkbox"/> <b>materias</b>	★	Examinar	✓ Estructura	🔍 Buscar	➕ Insertar	🗑 Vaciá	🚫 Eliminar
<input type="checkbox"/> <b>usuarios</b>	★	Examinar	✓ Estructura	🔍 Buscar	➕ Insertar	🗑 Vaciá	🚫 Eliminar
<b>3 tablas</b>	<b>Número de filas</b>						<b>0</b>

#### Ilustración 41.

## Tema 2: PHP - PDO

Volvamos hablar de PHP en 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.



**Ilustración 42.**

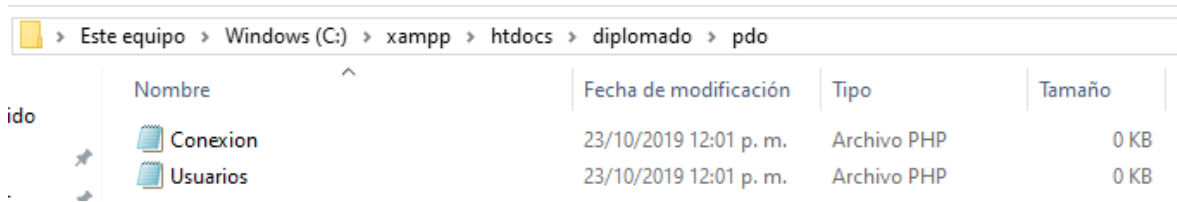
El sistema PDO se fundamenta en tres clases: PDO, PDOStatement y PDOException.

- 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:



Este equipo > Windows (C:) > xampp > htdocs > diplomado > pdo				
	Nombre	Fecha de modificación	Tipo	Tamaño
	Conexion	23/10/2019 12:01 p. m.	Archivo PHP	0 KB
	Usuarios	23/10/2019 12:01 p. m.	Archivo PHP	0 KB

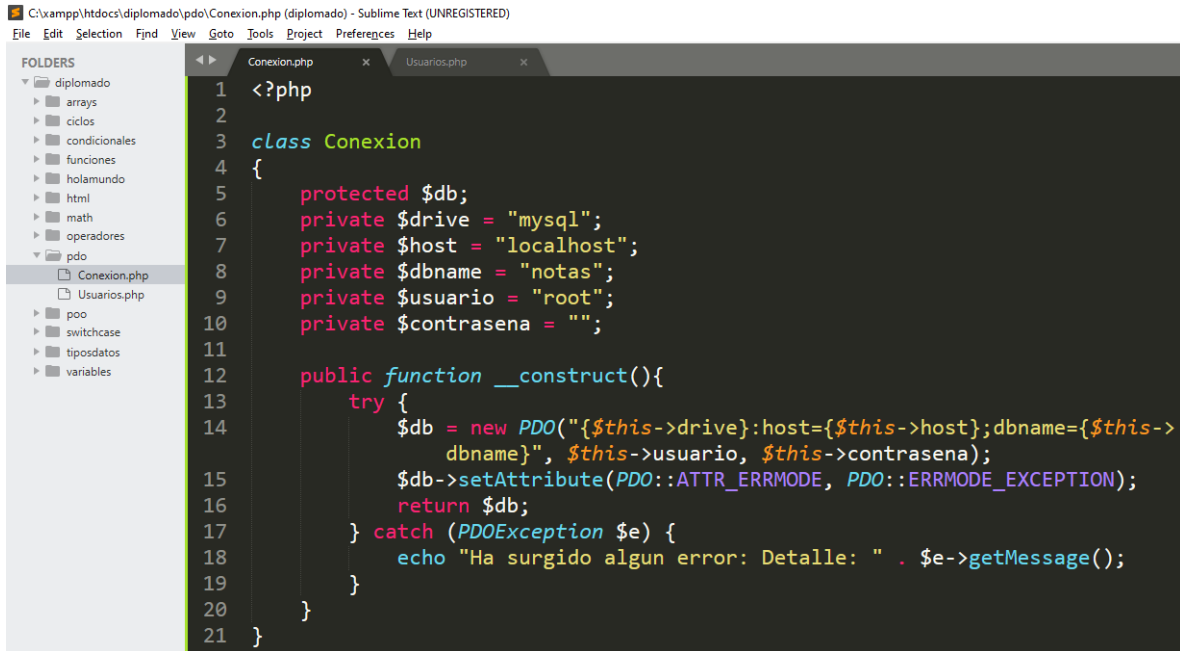
#### Ilustración 43.

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





```

1  <?php
2
3  class Conexion
4  {
5      protected $db;
6      private $drive = "mysql";
7      private $host = "localhost";
8      private $dbname = "notas";
9      private $usuario = "root";
10     private $contrasena = "";
11
12     public function __construct(){
13         try {
14             $db = new PDO("{ $this->drive}:host={ $this->host};dbname={ $this->
15                 dbname}", $this->usuario, $this->contrasena);
16             $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
17             return $db;
18         } catch (PDOException $e) {
19             echo "Ha surgido algun error: Detalle: " . $e->getMessage();
20         }
21     }
22 }

```

**Ilustración 44.**

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 = "";

```

**Ilustración 45.**

- **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, postgres, etc.
- **Private \$host = "localhost":** especifica el lugar donde se ejecutará la conexión, lo mismo ocurre con el lugar donde ejecutamos PHP:

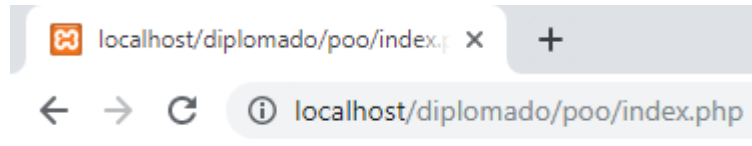


Ilustración 46.

- **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:

```
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();
    }
}
```

Ilustración 47.

- 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.

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

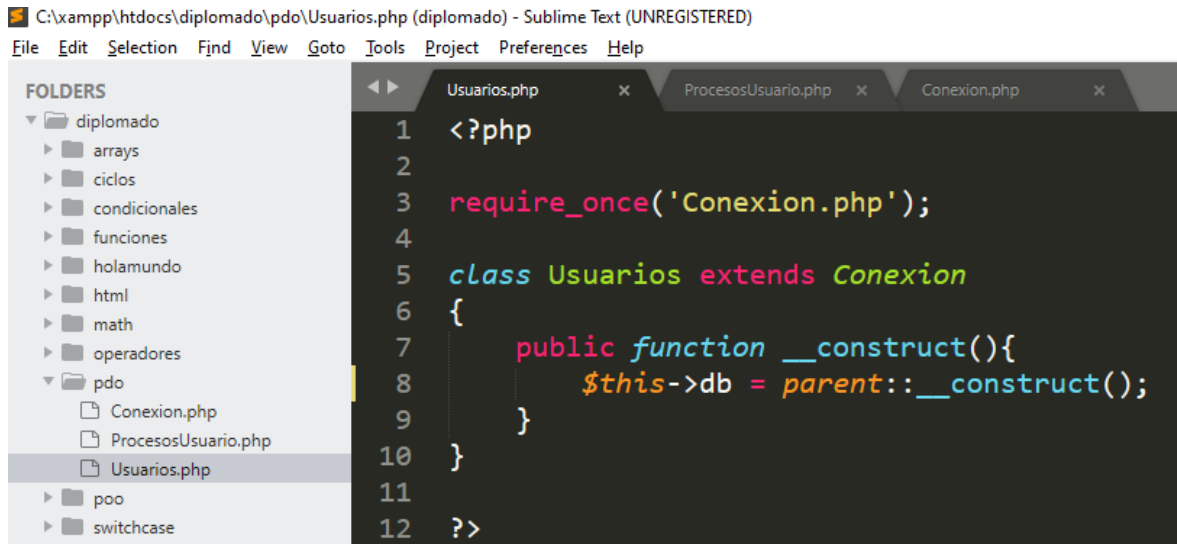
`PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION`.

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 *exception* 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.



C:\xampp\htdocs\diplomado\pdo\Usuarios.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

```

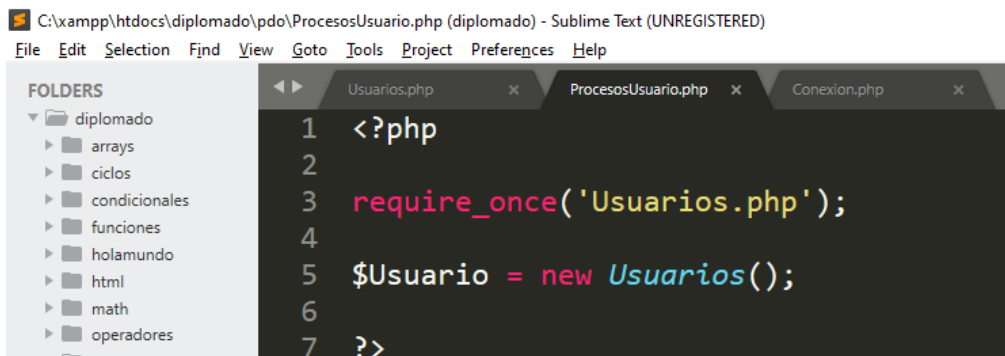
1  <?php
2
3  require_once('Conexion.php');
4
5  class Usuarios extends Conexion
6  {
7      public function __construct(){
8          $this->db = parent::__construct();
9      }
10 }
11
12 ?>

```

**Ilustración 48.**

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 esta. 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 creará un objeto.



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

```

1  <?php
2
3  require_once('Usuarios.php');
4
5  $Usuario = new Usuarios();
6
7  ?>

```

**Ilustración 49.**

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

```
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();
    }
}
```

Ilustración 50.

Un simple `echo` «Conectado.».

Ir al navegador, al archivo «ProcesosUsuario.php»:

<http://localhost/diplomado/pdo/ProcesosUsuario.php>

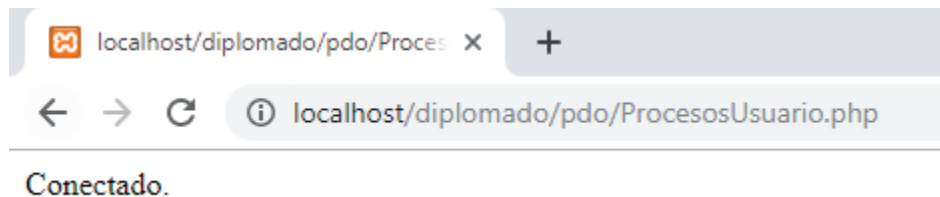


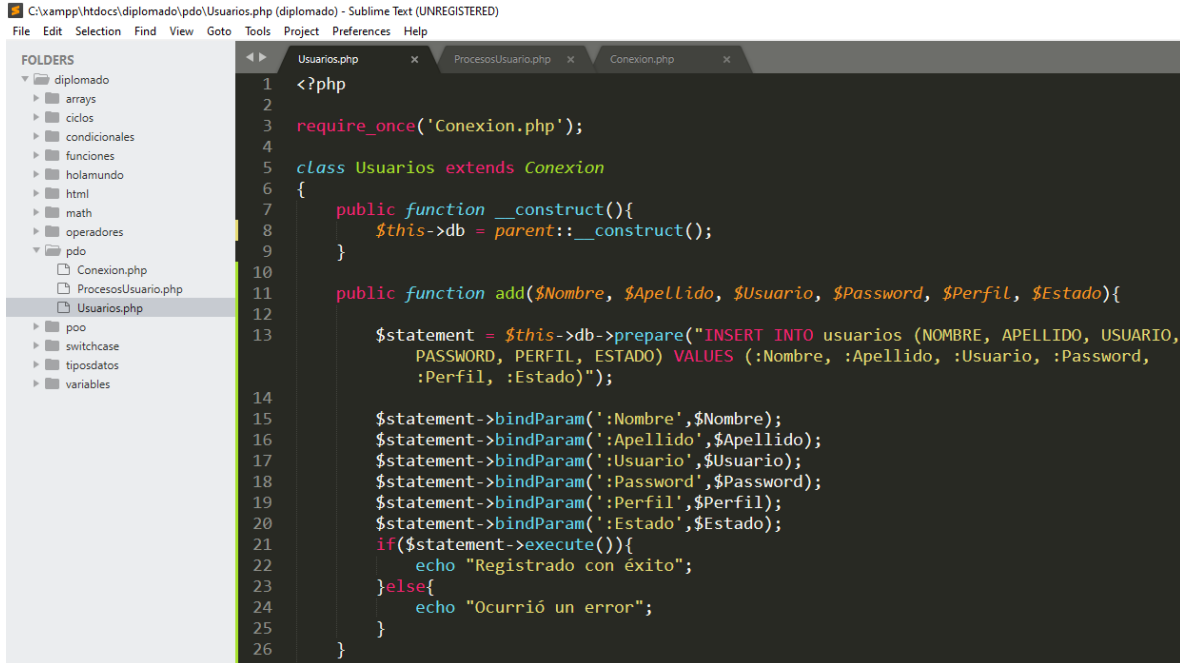
Ilustración 51.

Estaremos conectados exitosamente en nuestra 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:



```

1 <?php
2
3 require_once('Conexion.php');
4
5 class Usuarios extends Conexion
6 {
7     public function __construct(){
8         $this->db = parent::__construct();
9     }
10
11     public function add($Nombre, $Apellido, $Usuario, $Password, $Perfil, $Estado){
12
13         $statement = $this->db->prepare("INSERT INTO usuarios (NOMBRE, APELLIDO, USUARIO,
14             PASSWORD, PERFIL, ESTADO) VALUES (:Nombre, :Apellido, :Usuario, :Password,
15             :Perfil, :Estado)");
16
17         $statement->bindParam(':Nombre',$Nombre);
18         $statement->bindParam(':Apellido',$Apellido);
19         $statement->bindParam(':Usuario',$Usuario);
20         $statement->bindParam(':Password',$Password);
21         $statement->bindParam(':Perfil',$Perfil);
22         $statement->bindParam(':Estado',$Estado);
23         if($statement->execute()){
24             echo "Registrado con éxito";
25         }else{
26             echo "Ocurrió un error";
27         }
28     }
29 }

```

**Ilustración 52.**

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 descritos en nuestra tabla de la base de datos.

ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
------------	--------	----------	---------	----------	--------	--------

**Ilustración 53.**

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 realizar nuestra instrucción para insertar los datos. Recuerda que, al insertar datos en XAMPP, se tenían dos 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->prepare`) 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 seis campos de la tabla de nuestra base de datos, debemos referenciar seis

parámetros. Cada uno inicia con ":" y finaliza separando cada uno por ",". Ej.: (:Nombre, :Apellido, :Usuario, :Password, :Perfil, :Estado).

El resultado final es:

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

#### Ilustración 54.

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

Nuestro método recibe seis 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 **bindParam**, 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:

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

### Ilustración 55.

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

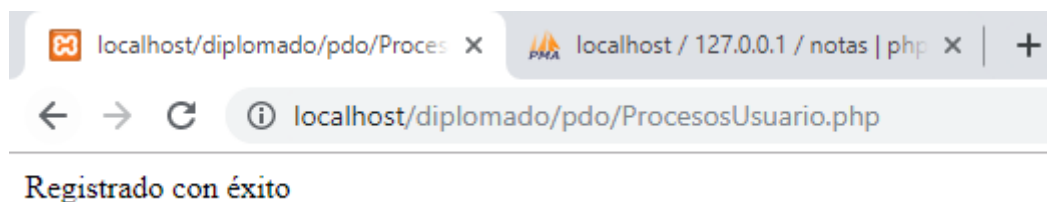
Finalmente, debemos ejecutar nuestra sentencia preparada por medio del método **execute()**, este método se encarga de ejecutar nuestra sentencia preparada y las referencias de los valores, en caso de que todo salga bien, retornará true, sino, false.

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

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

### Ilustración 56.

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] [Exportar]

☐ Mostrar todo | Número de filas: 25 | Filtrar filas:

+ Opciones

	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Diego	Palacio	dp11ego	123	Administrador	Activo

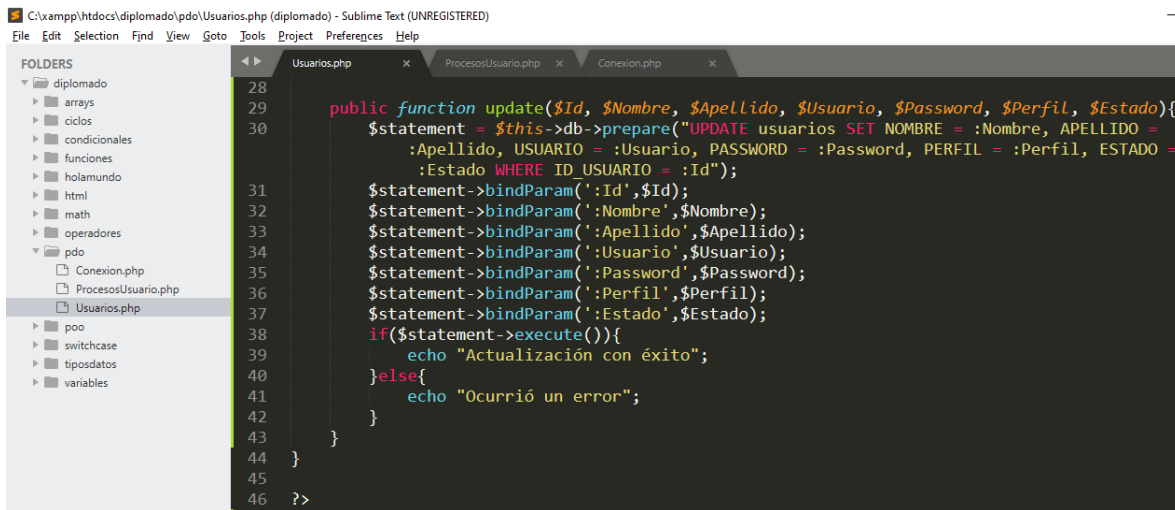
☐ Seleccionar todo | Para los elementos que están marcados: ☐ Editar ☐ Copiar ☐ Borrar ☐ Exportar

### Ilustraciones 57, 58 y 59.

**Actualizar.** La función actualizar es muy sencilla, parecida 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:



```

28
29
30 public function update($Id, $Nombre, $Apellido, $Usuario, $Password, $Perfil, $Estado){
    $statement = $this->db->prepare("UPDATE usuarios SET NOMBRE = :Nombre, APELLIDO =
    :Apellido, USUARIO = :Usuario, PASSWORD = :Password, PERFIL = :Perfil, ESTADO =
    :Estado WHERE ID_USUARIO = :Id");
31
32 $statement->bindParam(':Id',$Id);
33 $statement->bindParam(':Nombre',$Nombre);
34 $statement->bindParam(':Apellido',$Apellido);
35 $statement->bindParam(':Usuario',$Usuario);
36 $statement->bindParam(':Password',$Password);
37 $statement->bindParam(':Perfil',$Perfil);
38 $statement->bindParam(':Estado',$Estado);
39 if($statement->execute()){
40     echo "Actualización con éxito";
41 }else{
42     echo "Ocurrió un error";
43 }
44 }
45
46 ?>

```

### Ilustración 60.

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 **UPDATE** es que se realiza sobre registros ya insertados y que se encuentran en la base de datos. Cuando operamos con **UPDATE** 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 **UPDATE** determinar qué voy a actualizar y qué no voy a 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

#### Ilustración 61.

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, seis 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 cuáles datos quiero actualizar; si solo 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");
```

#### Ilustración 62.

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);
```

#### Ilustración 63.

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";
}
```

#### Ilustración 64.

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

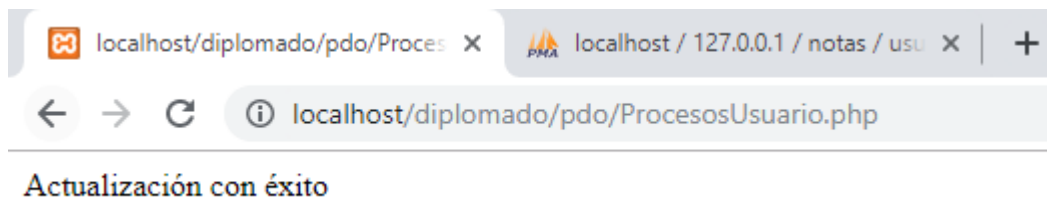
Opciones	ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Diego	Palacio	dp11ego	123	Administrador	Activo

#### Ilustración 65.

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:



Opciones		ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	1	Diego Alejandro	Palacio Valencia	dp11ego	abc9292	Docente	Activo

### Ilustraciones 66, 67 y 68.

El proceso funcionó correctamente.

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

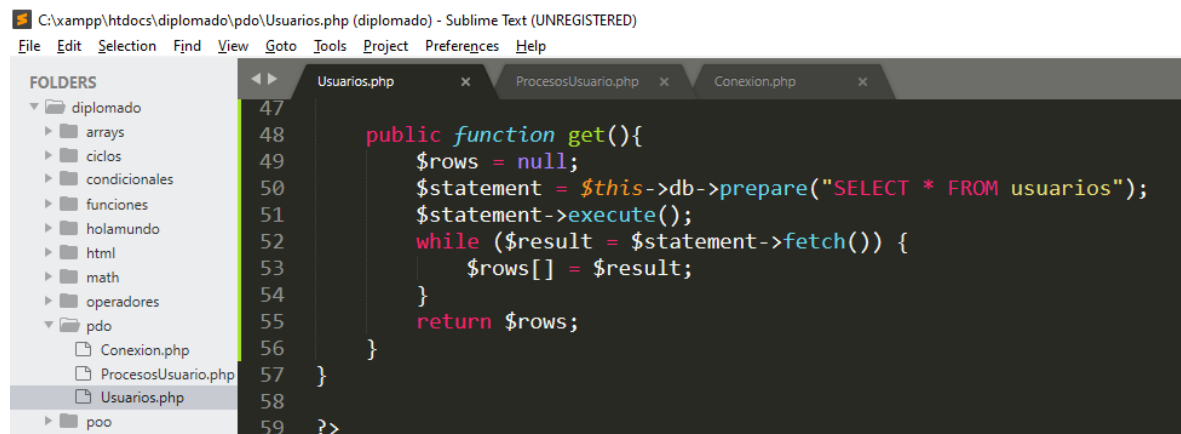
**Listar - Get.** La función listar es muy sencilla, esta permite consultar datos de una o varias tablas. Consta de un método que realizará uso de las clases de PDO para operar sobre las tablas de nuestra base de datos. Sintaxis:

*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:



```

C:\xampp\htdocs\diplomado\pdo\Usuarios.php (diplomado) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ diplomado
  └─ pdo
    └─ Usuarios.php

47
48 public function get(){
49     $rows = null;
50     $statement = $this->db->prepare("SELECT * FROM usuarios");
51     $statement->execute();
52     while ($result = $statement->fetch()) {
53         $rows[] = $result;
54     }
55     return $rows;
56 }
57
58
59 ?>
  
```

### Ilustración 69.

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 cláusula **WHERE** personalizada. Más adelante veremos esto.

La forma en que retornaremos los datos de nuestra tabla será por medio de un *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 **SELECT** 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 qué 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:

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

### Ilustración 70.

*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.

C:\xampp\htdocs\diplomado\pdo\ProcesosUsuario.php (diplomado) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

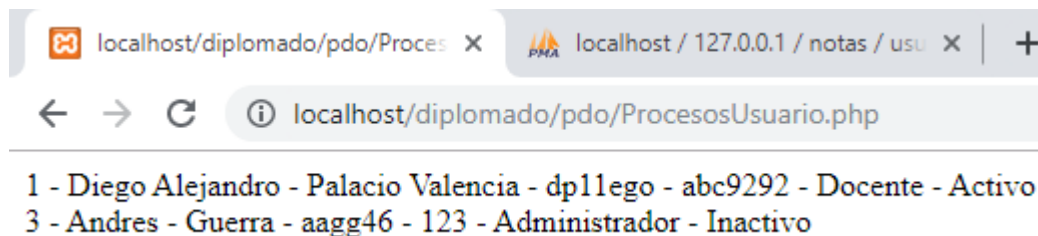
**FOLDERS**

- diplomado
  - arrays
  - ciclos
  - condicionales
  - funciones
  - holamundo
  - html
  - math
  - operadores
  - pdo
    - Conexion.php
    - ProcesosUsuario.php
    - Usuarios.php
  - poo
  - switchcase
  - tiposdatos
  - variables

```

1  <?php
2
3  require_once('Usuarios.php');
4
5  $Usuario = new Usuarios();
6
7  $getUsuarios = $Usuario->get();
8
9  foreach ($getUsuarios as $Usuario) {
10     echo $Usuario['ID_USUARIO'] . " - ";
11     echo $Usuario['NOMBRE'] . " - ";
12     echo $Usuario['APELLIDO'] . " - ";
13     echo $Usuario['USUARIO'] . " - ";
14     echo $Usuario['PASSWORD'] . " - ";
15     echo $Usuario['PERFIL'] . " - ";
16     echo $Usuario['ESTADO'] . "<br>";
17 }
18
19 ?>

```

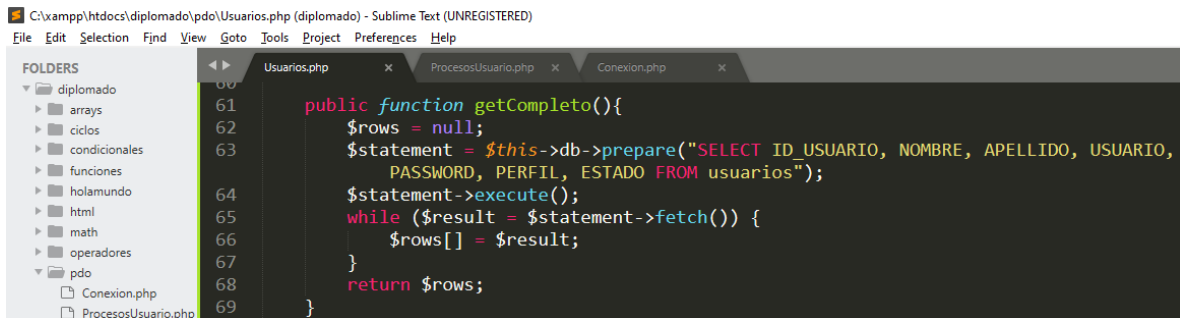


### Ilustraciones 71 y 72.

El proceso de impresión es bastante simple, es similar al visto en el módulo pasado con relación a los arreglos, simplemente realizaremos la impresión por

medio de un *foreach* y haciendo referencia a cada una de las columnas de nuestra tabla.

### Sintaxis 2:



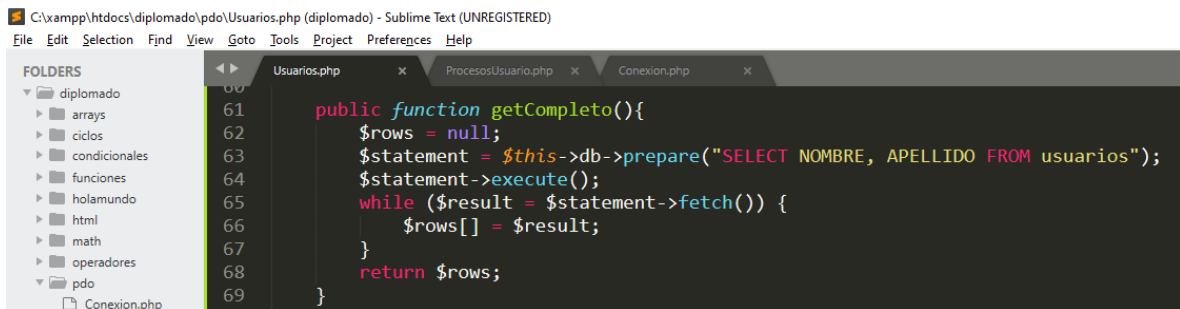
```
C:\xampp\htdocs\diplomado\pdo\Usuarios.php (diplomado) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ diplomado
  └─ arrays
  └─ ciclos
  └─ condicionales
  └─ funciones
  └─ holamundo
  └─ html
  └─ math
  └─ operadores
  └─ pdo
    └─ Conexion.php
    └─ ProcesosUsuario.php

61 public function getCompleto(){
62     $rows = null;
63     $statement = $this->db->prepare("SELECT ID_USUARIO, NOMBRE, APELLIDO, USUARIO,
64     PASSWORD, PERFIL, ESTADO FROM usuarios");
65     $statement->execute();
66     while ($result = $statement->fetch()) {
67         $rows[] = $result;
68     }
69     return $rows;
70 }
```

### Ilustración 73.

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\Usuarios.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

61 public function getCompleto(){
62     $rows = null;
63     $statement = $this->db->prepare("SELECT NOMBRE, APELLIDO FROM usuarios");
64     $statement->execute();
65     while ($result = $statement->fetch()) {
66         $rows[] = $result;
67     }
68     return $rows;
69 }
```

C:\xampp\htdocs\diplomado\pdo\ProcesosUsuario.php (diplomado) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

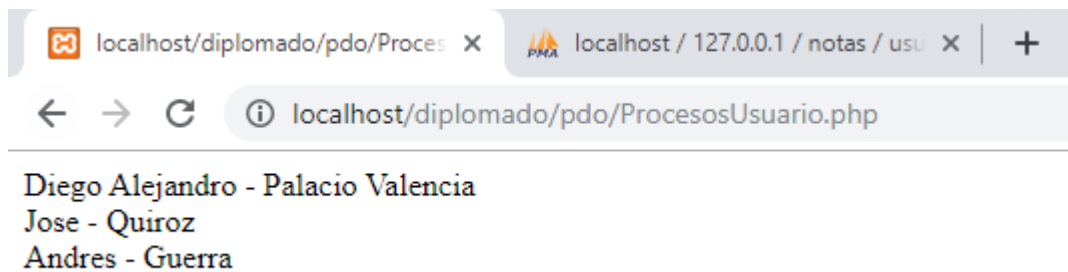
FOLDERS

- diplomado
  - arrays
  - ciclos
  - condicionales
  - funciones
  - holamundo
  - html
  - math
  - operadores
  - pdo
    - Conexion.php
    - ProcesosUsuario.php
    - Usuarios.php
  - poo
  - switchcase
  - tiempo

```

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 ?>

```

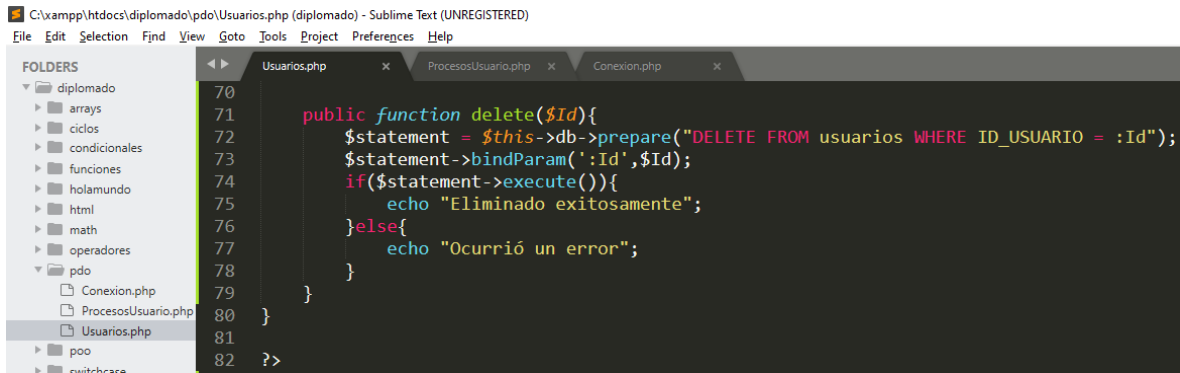


**Ilustraciones 74, 75 y 76.**

**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:

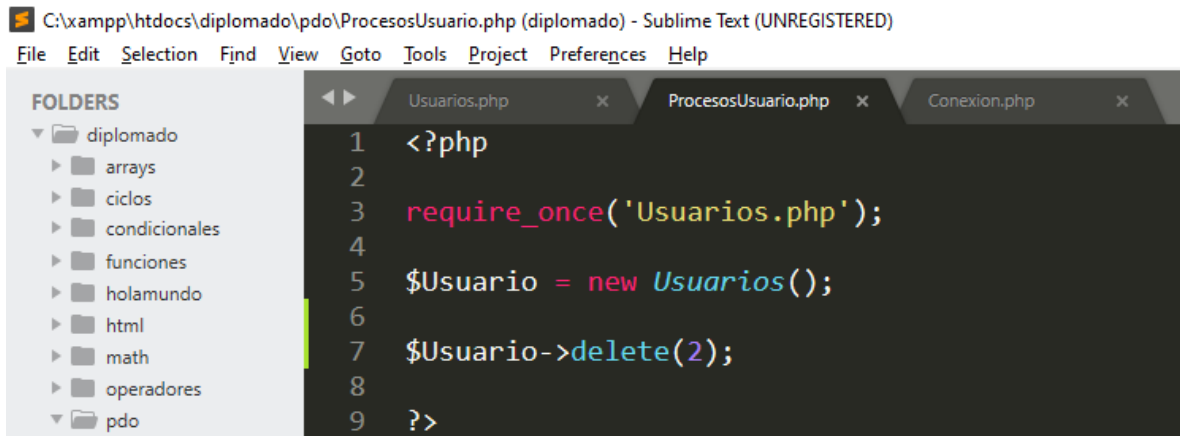


```

70
71
72 public function delete($Id){
73     $statement = $this->db->prepare("DELETE FROM usuarios WHERE ID_USUARIO = :Id");
74     $statement->bindParam(':Id',$Id);
75     if($statement->execute()){
76         echo "Eliminado exitosamente";
77     }else{
78         echo "Ocurrió un error";
79     }
80 }
81
82 ?>
    
```

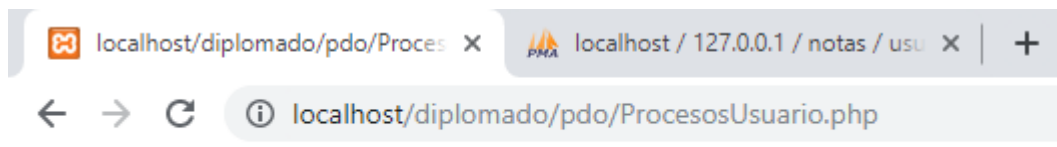
**Ilustración 77.**

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:



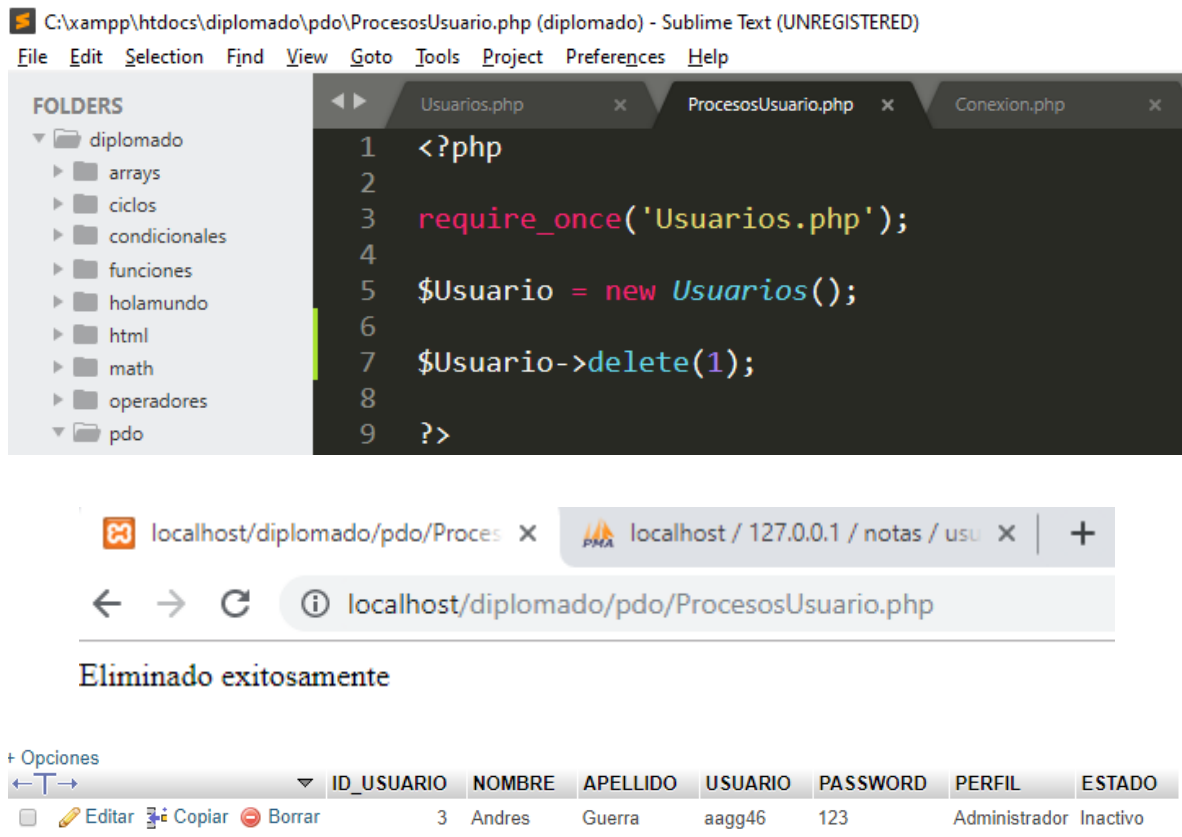
```

1 <?php
2
3 require_once('Usuarios.php');
4
5 $Usuario = new Usuarios();
6
7 $Usuario->delete(2);
8
9 ?>
    
```



**Eliminado exitosamente**

+ Opciones		ID_USUARIO	NOMBRE	APELLIDO	USUARIO	PASSWORD	PERFIL	ESTADO
<input type="checkbox"/>	Editar	1	Diego Alejandro	Palacio Valencia	dp11ego	abc9292	Docente	Activo
<input type="checkbox"/>	Editar	3	Andres	Guerra	aagg46	123	Administrador	Inactivo



Ilustraciones 78, 79, 80, 81, 82 y 83.

## Tema 3: Desarrollo del Proyecto - Estructura



Este apartado del diplomado se encuentra en video, puedes verlo y entender un poco más el proceso. Disponible en [https://www.youtube.com/watch?v=P7NQRut0X\\_w&feature=youtu.be](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 de este consiste en dividir el proyecto por MÓDULOS, es decir, cada posible entidad de nuestro proyecto se puede convertir en un MÓDULO, veamos:

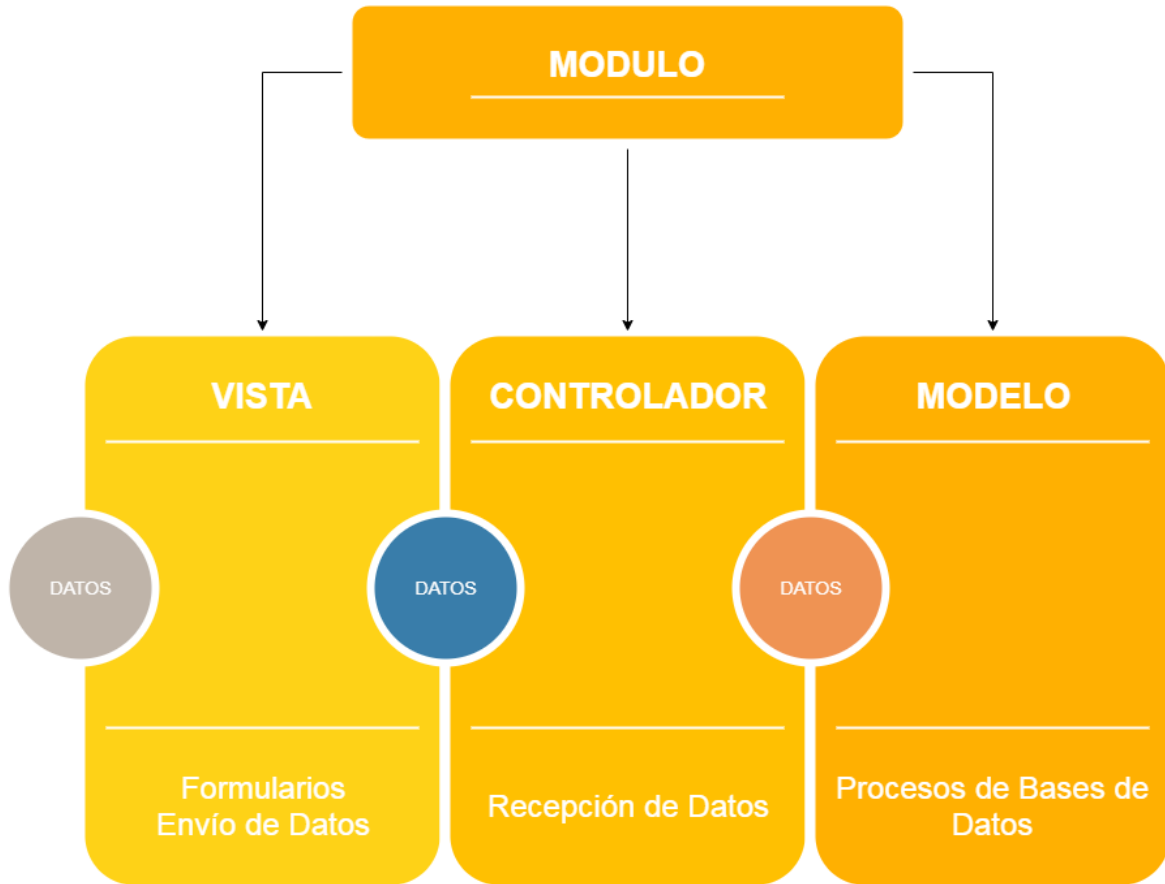
- Administradores.
- Docentes.
- Estudiantes.
- Materias.
- Usuarios.

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

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

La arquitectura MVC se divide en tres bloques fundamentales:

- Modelo.
- Vista.
- Controlador.



**Ilustración 84.**

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

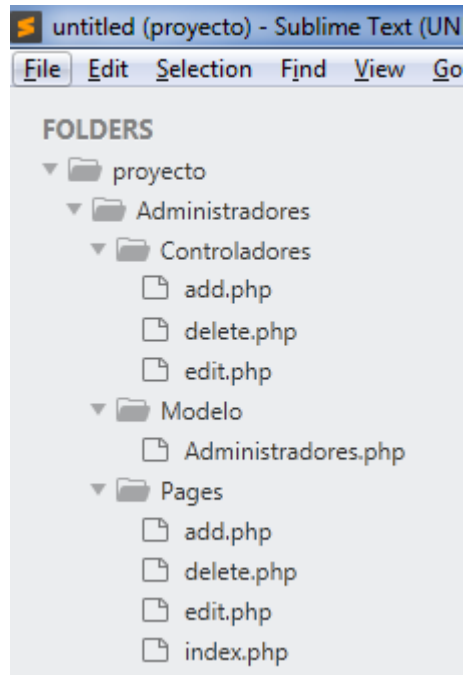
Esta es una forma muy general para no entrar en mucho detalle, veamos un poco cómo opera.





**Ilustración 85.**

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



**Ilustración 86.**

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

- Controladores.
- Modelo.
- Pages.

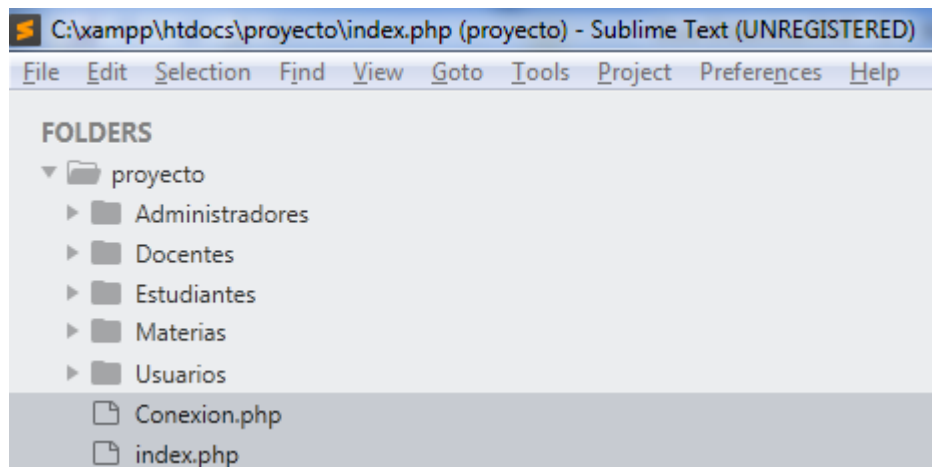
Y dentro de cada una de estas divisiones se encuentra el código fuente y los archivos respectivos.

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

Adicional a todo esto, hacen falta dos archivos importantes:

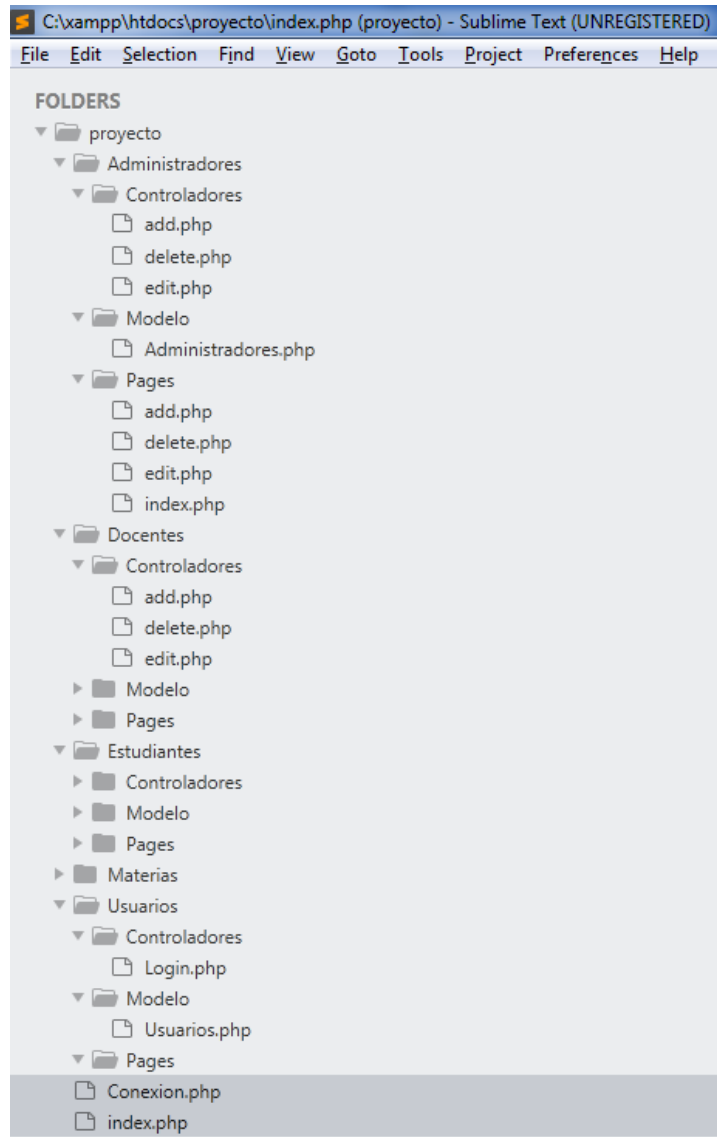
- Conexión.php
- Index.php

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



**Ilustración 87.**

Vea completo el esquema de nuestro proyecto:



### Ilustración 88.

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 video, puedes verlo y entender un poco más el proceso. Disponible en [https://www.youtube.com/watch?v=P7NQRut0X\\_w&feature=youtu.be](https://www.youtube.com/watch?v=P7NQRut0X_w&feature=youtu.be)

## Tema 5: Desarrollo del Proyecto - MVC



Este apartado del diplomado se encuentra en video, puedes verlo y entender un poco más el proceso. Disponible en [https://www.youtube.com/watch?v=tlfHz\\_RxiKE](https://www.youtube.com/watch?v=tlfHz_RxiKE)

## Tema 6: Desarrollo del Proyecto - Funcionalidades (I)



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

## Tema 7: Desarrollo del Proyecto - Funcionalidades (II)




Este apartado del diplomado se encuentra en video, puedes verlo y entender un poco más el proceso. Disponible en <https://www.youtube.com/watch?v=JV6UFxsu7uo>

## Tema 8: Desarrollo del Proyecto - Funcionalidades (III)



Este apartado del diplomado se encuentra en video, puedes verlo y entender un poco más el proceso. Disponible en [https://www.youtube.com/watch?v=\\_SVJP8BjAwE](https://www.youtube.com/watch?v=_SVJP8BjAwE)





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, 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.

## **GUÍA DIDÁCTICA 5**

M2-DV59-GU05

MÓDULO 5: DESARROLLO WEB II

© DERECHOS RESERVADOS - POLITÉCNICO DE COLOMBIA, 2023  
Medellín, Colombia

Proceso: Gestión Académica Virtual  
Realización del texto: Diego Palacio, docente  
Revisión del texto: Comité de Revisión  
Diseño: Comunicaciones

Editado por el Politécnico de Colombia.