



PHP. Nivel III

mayo, 2019



Objetivos del nivel

- Aprender a crear bases de datos MySQL
- Conectarse a una base de datos MySQL
- Consultar y manipular los datos de la base de datos
- Aprender a crear gráficos

Prerrequisitos del nivel

- PHP Nivel II
- Bases de Datos Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADIF1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2019. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Manejo de Errores

- 1.1.- Configuración de Reporte de Errores.
- 1.2.- Configuración en Tiempo de Ejecución.
- 1.3.- Operador de Control de Errores.

Capítulo 2. Manejo de Excepciones

- 2.1.- Thrown.
- 2.2.- Try Catch.
- 2.3.- Clase Exception.

Capítulo 3. Phpmyadmin

- 3.1.- Mysql y Phpmyadmin.
- 3.2.- Crear Una Base de Datos.
- 3.3.- Crear Tablas.
- 3.4.- Manipular Los Datos.
- 3.5.- Exportar e Importar Bases de Datos.

Capítulo 4. Php Data Object

- 4.1.- Introducción.
- 4.2.- Conexión a Base de Datos.
- 4.3.- Errores de Conexión.

Capítulo 5. Consultar Información

- 5.1.- Ejecutar Consultas.
- 5.2.- Acceder a Los Registros.
- 5.3.- Acceder a Los Campos.

Capítulo 6. Buscar Registros

- 6.1.- Formulario.

6.2.- Procesar la Búsqueda.

Capítulo 7. Listados

7.1.- Llenar Select.

7.2.- La Propiedad Value.

Capítulo 8. Incluir Registros

8.1.- Incluir Registros.

8.2.- Validación.

Capítulo 9. Modificar Registros

9.1.- Modificar un Registro.

Capítulo 10. Eliminar

10.1.- Eliminar un Registro.

Capítulo 11. Eliminar Varios Registros

11.1.- Introducción.

11.2.- Formulario.

11.3.- Procesar Los Elementos.



Capítulo 12. Instrucciones Preparadas

12.1.- Instrucciones Preparadas.

12.2.- Parámetros en Instrucciones Preparadas.

Capítulo 1. MANEJO DE ERRORES

1.1.- Configuración de Reporte de Errores

PHP posee 2 parámetros de configuración que determinan el comportamiento al momento de ocurrir errores en tiempo de ejecución:

- `Display_errors`: establece si los errores se mostrarán.
- `Error_reporting`: establece cuales errores de PHP son notificados en tiempo de ejecución. PHP tiene varios niveles de errores para notificar.

El valor de ambos parámetros se establece de forma global para todas las aplicaciones en el archivo `php.ini`. En la siguiente imagen se muestra un ejemplo de la sección de configuración de errores en un archivo `PHP.ini`:



The screenshot shows a Windows Notepad window titled "php.ini - Notepad". The content of the file is a configuration section for error reporting. It starts with a semi-colon and the text "Error Level Constants:" followed by a list of constants and their descriptions. The constants listed are E_ALL, E_ERROR, E_RECOVERABLE_ERROR, E_WARNING, E_PARSE, E_NOTICE, E_STRICT, E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR, E_COMPILE_WARNING, E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE, E_DEPRECATED, and E_USER_DEPRECATED. Below this, there is a section titled "Common Values:" which contains several pre-defined combinations of error levels separated by pipe symbols (|). At the bottom of the section, it specifies the "Default Value" as E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED, the "Development Value" as E_ALL, and the "Production Value" as E_ALL & ~E_DEPRECATED & ~E_STRICT. The URL "http://php.net/error-reporting" is also mentioned at the bottom. The Notepad window has a blue header bar and a white body with black text.

```
; Error Level Constants:  
; E_ALL           - All errors and warnings (includes E_STRICT as of PHP 5.4.0)  
; E_ERROR          - fatal run-time errors  
; E_RECOVERABLE_ERROR - almost fatal run-time errors  
; E_WARNING        - run-time warnings (non-fatal errors)  
; E_PARSE           - compile-time parse errors  
; E_NOTICE          - run-time notices (these are warnings which often result  
;                      from a bug in your code, but it's possible that it was  
;                      intentional (e.g., using an uninitialized variable and  
;                      relying on the fact it is automatically initialized to an  
;                      empty string))  
; E_STRICT          - run-time notices, enable to have PHP suggest changes  
;                      to your code which will ensure the best interoperability  
;                      and forward compatibility of your code  
; E_CORE_ERROR     - fatal errors that occur during PHP's initial startup  
; E_CORE_WARNING   - warnings (non-fatal errors) that occur during PHP's  
;                      initial startup  
; E_COMPILE_ERROR  - fatal compile-time errors  
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)  
; E_USER_ERROR     - user-generated error message  
; E_USER_WARNING   - user-generated warning message  
; E_USER_NOTICE    - user-generated notice message  
; E_DEPRECATED     - warn about code that will not work in future versions  
;                      of PHP  
; E_USER_DEPRECATED - user-generated deprecation warnings  
  
; Common Values:  
; E_ALL (Show all errors, warnings and notices including coding standards.)  
; E_ALL & ~E_NOTICE (Show all errors, except for notices)  
; E_ALL & ~E_NOTICE & ~E_STRICT (Show all errors, except for notices and coding standards warnings.)  
; E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR (Show only errors)  
; Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED  
; Development Value: E_ALL  
; Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT  
; http://php.net/error-reporting  
error_reporting = E_ALL
```

Los posibles valores que puede tener `error_reporting` son los siguientes:

| Valor | Constante | Descripción |
|-------|--|---|
| 1 | <code>E_ERROR</code> (integer) | Errores Fatales en tiempo de ejecución. Éstos indican errores que no se pueden recuperar, tales como un problema de asignación de memoria. La ejecución del script se interrumpe. |
| 2 | <code>E_WARNING</code> (integer) | Advertencias en tiempo de ejecución (errores no fatales). La ejecución del script no se interrumpe. |
| 4 | <code>E_PARSE</code> (integer) | Errores de análisis en tiempo de compilación. Los errores de análisis deberían ser generados únicamente por el analizador. |
| 8 | <code>E_NOTICE</code> (integer) | Avisos en tiempo de ejecución. Indican que el script encontró algo que podría señalar un error, pero que también podría ocurrir en el curso normal al ejecutar un script. |
| 16 | <code>E_CORE_ERROR</code> (integer) | Errores fatales que ocurren durante el arranque inicial de PHP. Son como un <code>E_ERROR</code> , excepto que son generados por el núcleo de PHP. |
| 32 | <code>E_CORE_WARNING</code> (integer) | Advertencias (errores no fatales) que ocurren durante el arranque inicial de PHP. Son como un <code>E_WARNING</code> , excepto que son generados por el núcleo de PHP. |
| 64 | <code>E_COMPILE_ERROR</code> (integer) | Errores fatales en tiempo de compilación. Son como un <code>E_ERROR</code> , excepto que son generados por Motor de Script Zend. |

Otros valores:

| Valor | Constante | Descripción |
|-------|---------------------------------------|--|
| 64 | E_COMPILE_ERROR (integer) | Errores fatales en tiempo de compilación. Son como un E_ERROR , excepto que son generados por Motor de Script Zend. |
| 128 | E_COMPILE_WARNING (integer) | Advertencias en tiempo de compilación (errores no fatales). Son como un E_WARNING , excepto que son generados por Motor de Script Zend. |
| 256 | E_USER_ERROR (integer) | Mensaje de error generado por el usuario. Es como un E_ERROR , excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error() . |
| 512 | E_USER_WARNING (integer) | Mensaje de advertencia generado por el usuario. Es como un E_WARNING , excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error() . |
| 1024 | E_USER_NOTICE (integer) | Mensaje de aviso generado por el usuario. Es como un E_NOTICE , excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error() . |
| 2048 | E_STRICT (integer) | Habilítelo para que PHP sugiera cambios en su código, lo que asegurará la mejor interoperabilidad y compatibilidad con versiones posteriores de PHP de su código. |

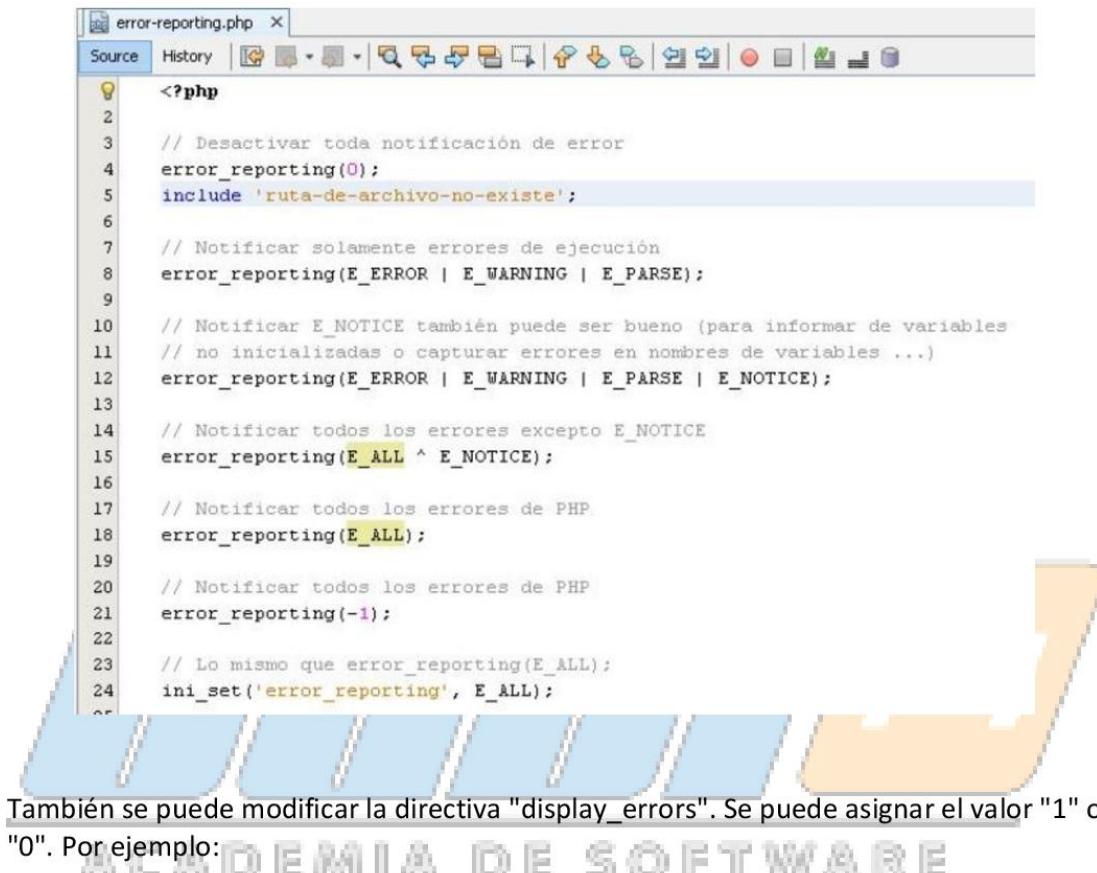
Otros valores:

| Valor | Constante | Descripción |
|-------|--|---|
| 2048 | E_STRICT (integer) | Habilítelo para que PHP sugiera cambios en su código, lo que asegurará la mejor interoperabilidad y compatibilidad con versiones posteriores de PHP de su código. |
| 4096 | E_RECOVERABLE_ERROR (integer) | Error fatal capturable. Indica que ocurrió un error probablemente peligroso, pero no dejó al Motor en un estado inestable. Si no se captura el error mediante un gestor definido por el usuario (vea también set_error_handler()), la aplicación se abortará como si fuera un E_ERROR . |
| 8192 | E_DEPRECATED (integer) | Avisos en tiempo de ejecución. Habilítelo para recibir avisos sobre código que no funcionará en futuras versiones. |
| 16384 | E_USER_DEPRECATED (integer) | Mensajes de advertencia generados por el usuario. Son como un E_DEPRECATED , excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error() . |
| 32767 | E_ALL (integer) | Todos los errores y advertencias soportados, excepto del nivel E_STRICT antes de PHP 5.4.0. |

1.2.- Configuración en Tiempo de Ejecución

Los valores de estas 2 directivas se pueden modificar en tiempo de ejecución. Con la función "error_reporting" se puede cambiar específicamente el tipo de errores que se van a mostrar.

Con el uso de la función ini_set se pueden modificar los valores de cualquier directiva. Por ejemplo:



```
<?php
// Desactivar toda notificación de error
error_reporting(0);
include 'ruta-de-archivo-no-existe';

// Notificar solamente errores de ejecución
error_reporting(E_ERROR | E_WARNING | E_PARSE);

// Notificar E_NOTICE también puede ser bueno (para informar de variables
// no inicializadas o capturar errores en nombres de variables ...)
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Notificar todos los errores excepto E_NOTICE
error_reporting(E_ALL ^ E_NOTICE);

// Notificar todos los errores de PHP
error_reporting(E_ALL);

// Notificar todos los errores de PHP
error_reporting(-1);

// Lo mismo que error_reporting(E_ALL);
ini_set('error_reporting', E_ALL);
```

También se puede modificar la directiva "display_errors". Se puede asignar el valor "1" o "0". Por ejemplo:



```
<?php
error_reporting(E_ALL);
ini_set("display_errors", 0);
include 'ruta-de-archivo-no-existe'; // no mostrara los errores

ini_set("display_errors", 1);
include 'ruta-de-archivo-no-existe'; // mostrara los errores
```

1.3.- Operador de Control de Errores

PHP soporta un operador de control de errores: el signo de arroba (@). Cuando se antepone a una expresión en PHP, cualquier mensaje de error que pueda ser generado por esa expresión será ignorado. Por ejemplo:

The screenshot shows a code editor interface with a toolbar at the top. The code in the editor is as follows:

```
<?php  
1  
2 @include 'ruta-de-archivo-no-existe'; // no muestra el error si no encuentra el archivo  
3  
4  
5 $usuarios = Array(  
6  
7     "Administrador" => "Pedro",  
8     "Empleado"      => "Maria"  
9 );  
10  
11 $nombre = @$usuarios["Gerente"]; // no mostrara el error aunque la clave no exista  
12  
13 $nombre = $usuarios["Gerente"]; // mostrara error porque la clave no existe  
14  
15  
16
```

The browser address bar shows "localhost/practica-errores/operador-control-errores.php". Below the browser window, a notice message is displayed: "Notice: Undefined index: usuario in C:\xampp\htdocs\practica-errores\operador-control-errores.php on line 14".

El operador @ trabaja solo sobre expresiones. Una simple regla de oro es: si se puede tomar el valor de algo, entonces se le puede anteponer el operador @. Por ejemplo, puede anteponerse a variables, a llamadas a funciones e includes, constantes y así sucesivamente. No puede anteponerse a definiciones de función o clase, ni a estructuras condicionales como if, foreach y así sucesivamente.

Capítulo 2. MANEJO DE EXCEPCIONES

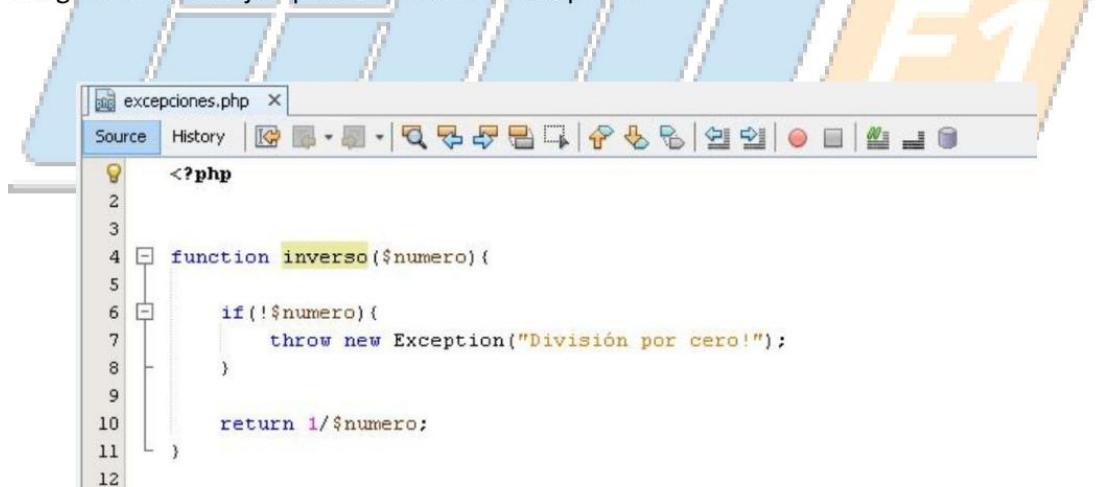
2.1.- Thrown

A partir de PHP 5 se incorporó una nueva forma orientada a objetos de manejar los errores. Las excepciones se utilizan para cambiar el flujo normal de un script si ocurre un error concreto dentro de una condición. Esta condición es lo que se denomina excepción.

Estas excepciones tienen un modelo similar al de otros lenguajes de programación. Una excepción puede ser lanzada ("thrown"), y atrapada ("caught") dentro de PHP. El código puede estar dentro de un bloque try para facilitar la captura de excepciones potenciales. Cada bloque try debe tener al menos un bloque catch o finally correspondiente.

El objeto lanzado debe ser una instancia de la clase de Exception o una subclase de Exception. Intentar lanzar un objeto que no lo sea resultara en un error fatal de PHP.

El siguiente es un ejemplo de lanzar una excepción:



The screenshot shows a code editor window titled "excepciones.php". The code is as follows:

```
<?php
function inverso($numero) {
    if (!$numero) {
        throw new Exception("División por cero!");
    }
    return 1/$numero;
}
```

2.2.- Try Catch

Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado y PHP intentará encontrar un bloque "catch" coincidente para manejar la excepción. Si la excepción no es capturada, se emitirá un error fatal de PHP con un mensaje "Uncaught

Exception..." ("Excepción no capturada"), a menos que se haya definido un manejador con `set_exception_handler()`. El siguiente es un ejemplo de como capturar una excepción:

The screenshot shows a code editor window with the file 'excepciones.php' open. The code defines a function `inverso` that throws an exception if the input is zero. It then attempts to call this function with arguments 2 and 0, and catches the exception to echo its message.

```
<?php
function inverso($numero){
    if(!$numero){
        throw new Exception("División por cero!");
    }
    return 1/$numero;
}
// el siguiente código emitirá una excepción con éxito!
try{
    inverso(2) . "\n";
    inverso(0) . "\n";
} catch (Exception $ex) {
    echo 'Excepción capturada: ', $ex->getMessage(), "\n";
}
```

The browser status bar at the bottom shows the URL `localhost/practica-errores/excepciones.php`. The page content displays the captured exception message: `Excepción capturada: División por cero!`.

Si no se captura la excepción:

The screenshot shows a code editor window titled "excepciones.php". The code is as follows:

```
<?php  
2  
3  
4     function inverso($numero){  
5         if(!$numero){  
6             throw new Exception("División por cero!");  
7         }  
8         return 1/$numero;  
9     }  
10    // el siguiente código emitirá un error fatal  
11    inverso(0);  
12  
13  
14  
15  
16
```

A yellow lightning bolt icon is positioned to the right of the code editor. Below the editor, a message box displays the error:

Fatal error: Uncaught Exception: División por cero! in C:\xampp\htdocs\practica-errores\excepciones.php on line 7

2.3.- Clase Exception

Una clase de excepción definida por el usuario puede ser definida ampliando de la clase Exception interna. Si una clase extiende de la clase interna Exception y redefine el constructor, se recomienda encarecidamente que también llame a parent::__construct(); para asegurarse que toda la información disponible haya sido asignada apropiadamente. Por ejemplo:

```
<?php

2
3 class MiException extends Exception {
4
5     public function MensajeError() {
6         // Mensaje de error
7         $error = 'Error: '
8         |.$this->getMessage().' en la linea '
9         .$.this->getLine().' en el archivo '
10        .$.this->getFile();
11
12        return $error;
13    }
}
```

El siguiente es un ejemplo de como lanzar y capturar una excepción personalizada:



The screenshot shows a Java IDE interface with a PHP file open. The code defines a function `inverso` that calculates the reciprocal of a number. It includes a try-catch block that catches `MiException` and prints the error message. The code is annotated with numbers 1 through 22.

```
<?php
1 include "excepcion-personalizada.php";
2
3 function inverso($numero){
4
5     if(!$numero){
6         throw new MiException("División por cero!");
7     }
8
9     return 1/$numero;
10
11 }
12
13 // el siguiente código emitirá una excepción con éxito!
14
15 try{
16     inverso(2) . "\n";
17     inverso(0). "\n";
18
19 } catch (MiException $ex) {
20
21     echo $ex->MensajeError();
22 }
```

Capítulo 3. PHPMYADMIN

3.1.- Mysql y Phpmyadmin

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado, lo ofrece bajo la GNU GPL, pero, empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia que les permita ese uso. Está desarrollado en su mayor parte en ANSI C.

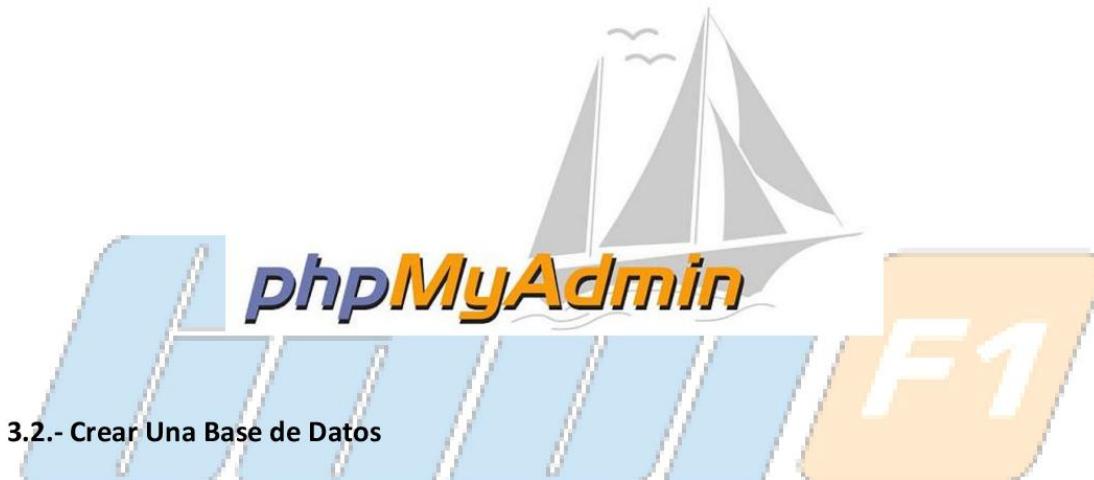
MySQL funciona sobre múltiples plataformas, incluyendo AIX, BSD, FreeBSD, HP-UX, GNU/Linux, Mac OS X, NetBSD, Novell Netware, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Vista y otras versiones de Windows.



Existe una gran cantidad de programas que sirven de "Fronts" de MySQL, entre los mas populares están: MysqlFront, MysqlCC, PhpMyAdmin y MySQL Workbench. El Front que se utilice dependerá básicamente de del gusto del programador, o de la amigabilidad y funcionalidad del software.

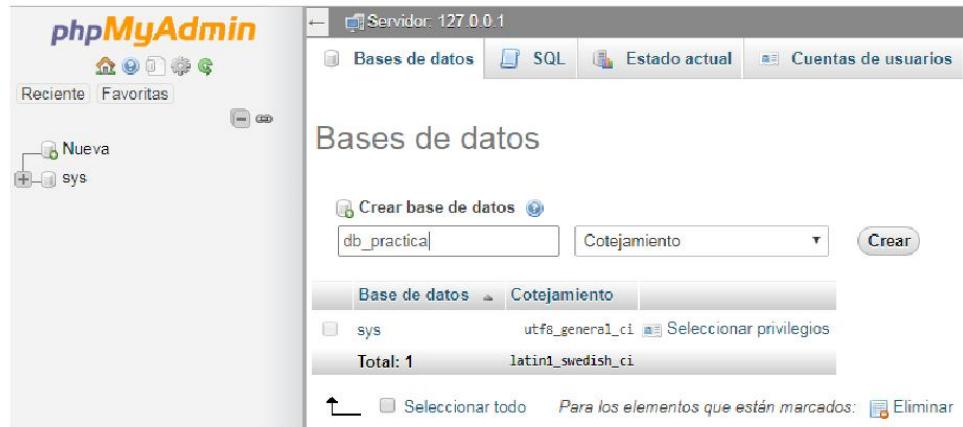
PhpMyAdmin es una herramienta cuya característica distintiva de los demás es que posee una interfaz web. Para conectarse al servidor local usando phpMyAdmin se debe abrir un navegador Web y colocar en la barra de direcciones:

[http://localhost/phpmyadmin.](http://localhost/phpmyadmin)



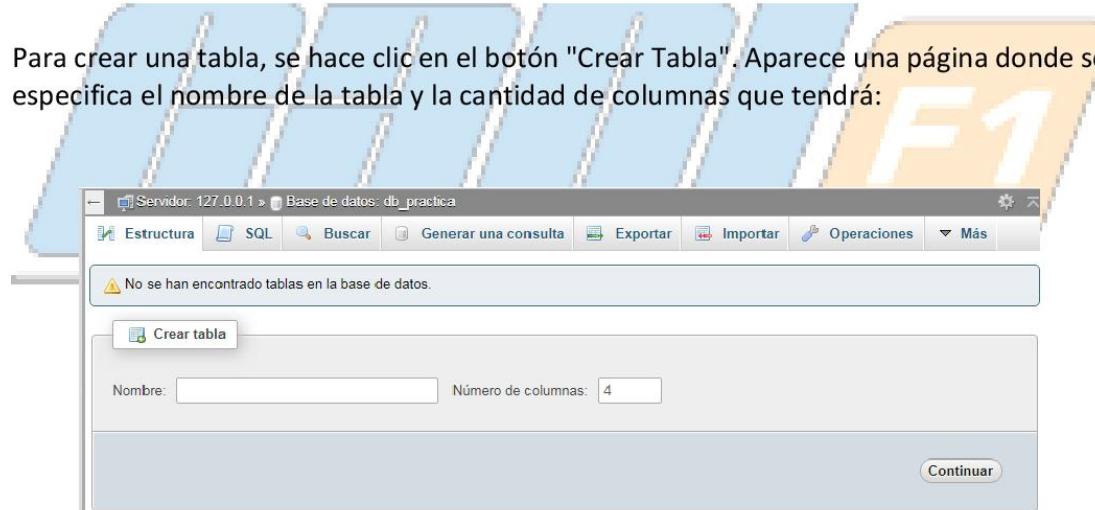
A screenshot of the "Configuraciones generales" (General Configuration) page in PhpMyAdmin. The page title is "Configuraciones generales". It contains a single configuration option: "Cotejamiento de la conexión al servidor" (Check connection to server) with a dropdown menu showing "utf8mb4_general_ci". The left sidebar shows a tree structure with "Nueva" selected under "Bases de datos".

Se establece el nombre de la nueva base de datos y se hace click en el botón "Crear":



3.3.- Crear Tablas

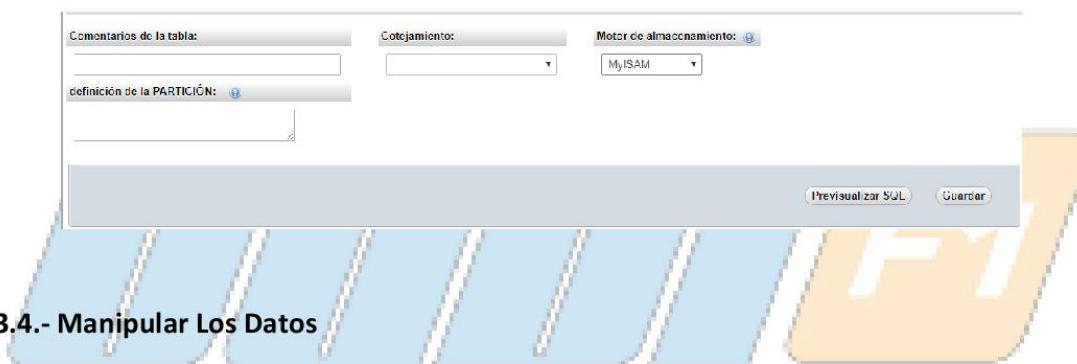
Para crear una tabla, se hace clic en el botón "Crear Tabla". Aparece una página donde se especifica el nombre de la tabla y la cantidad de columnas que tendrá:



Luego aparece una página donde se debe especificar los nombres de los campos con sus respectivos tipos de datos:



Al finalizar, se hace click en el botón "Guardar":



3.4.- Manipular Los Datos

Después de crear la tabla se pueden agregar registros, buscarlos, editarlos o eliminarlos.



Al hacer click en el hipervínculo "Insertar" aparecerá una página para darle valor a cada uno de los campos del nuevo registro:

| Columna | Tipo | Función | Nulo | Valor |
|---------|------------------|---------|------|-----------|
| cedula | int(11) unsigned | | | 123456789 |
| nombre | varchar(255) | | | Juan |

Se pueden agregar varios capítulos a la vez:

| Columna | Tipo | Función | Nulo | Valor |
|---------|------------------|---------|------|---------|
| cedula | int(11) unsigned | | | 2456676 |
| nombre | varchar(255) | | | Maria |

| Columna | Tipo | Función | Nulo | Valor |
|---------|------------------|---------|------|----------|
| cedula | int(11) unsigned | | | 26554321 |
| nombre | varchar(255) | | | Aura |

Insertar como una nueva fila Volver

Al guardar, se muestra la instrucción SQL que se ejecuta:

Servidor: 127.0.0.1 > Base de datos: db_practica > Tabla: tb_prueba

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Más

2 filas insertadas.

```
INSERT INTO `tb_prueba` (`cedula`, `nombre`) VALUES ('2456676', 'Maria'), ('26554321', 'Aura');
```

[Editar en línea] [Editar] [Crear código PHP]

3.5.- Exportar e Importar Bases de Datos

En muchas ocasiones se necesita mover una base de datos de un servidor a otro (una computadora a otra). Esta tarea es muy común sobre todo a la hora de subir un sitio web a un hosting que use una base de datos MySQL. También a la hora de hacer respaldos de las bases de datos. La idea es generar un simple archivo de texto con un conjunto de instrucciones SQL necesarias para crear la base de datos.

A este archivo se le llama comúnmente "script de la base de datos". La forma de generar este script depende del front usado para administrar la base de datos. La mayoría de ellos está basado en una herramienta llamada "mysqldump" que está incluida en el paquete MySQL.

Se hace click en la opción "Exportar":

ACADEMIA DE SOFTWARE

Servidor: 127.0.0.1 > Base de datos: db_practica

Estructura SQL Buscar Generar una consulta Exportar Importar

Exportando tablas de la base de datos "db_practica"

Método de exportación:

- Rápido - mostrar sólo el mínimo de opciones de configuración
- Personalizado - mostrar todas las opciones de configuración posibles

Formato:

SQL

Continuar

Se genera un archivo descargable, que luego puede abrirse con un editor de texto:



El archivo generado es un simple archivo de texto que puede visualizarse con cualquier editor de texto, como: bloc de notas, word pad, notepad++, etc. Al abrir un archivo de script de base de datos deberíamos ver algo como esto:

The screenshot shows a Microsoft WordPad window titled "New Project 20090314 1733 - WordPad". The window contains a MySQL database creation script. The code includes comments for setting session variables, creating a schema, dropping and creating a table, and dumping data for that table. The code is as follows:

```
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNI  
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_C CHECKS  
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_ */;  
  
--  
-- Create schema phpmyadmin  
--  
  
CREATE DATABASE IF NOT EXISTS phpmyadmin;  
USE phpmyadmin;  
  
DROP TABLE IF EXISTS `pma_bookmark`;  
CREATE TABLE `pma_bookmark` (  
    `id` int(11) NOT NULL auto_increment,  
    `dbase` varchar(255) collate utf8_bin NOT NULL def  
    `user` varchar(255) collate utf8_bin NOT NULL defa  
  
--  
-- Dumping data for table `pma_bookmark`  
--  
  
/*!40000 ALTER TABLE `pma_bookmark` DISABLE KEYS */;
```

Importar la base de datos se refiere al proceso contrario al explicado en el paso anterior. La idea es crear una base de datos nueva (probablemente con todo y la información) a partir de un archivo de script ya existente, que fue exportado previamente en otro servidor MySQL.

The screenshot shows the 'Importar' (Import) page of the phpMyAdmin interface. The top navigation bar includes 'Servidor: 127.0.0.1', 'Base de datos: db_practica', and tabs for 'Estructura', 'SQL', 'Buscar', 'Generar una consulta', 'Exportar', 'Importar' (which is selected), 'Operaciones', and 'Más'. The main content area has a title 'Importando en la base de datos "db_practica"'.

Archivo a importar:
El archivo puede ser comprimido (gzip, zip) o descomprimido.
Un archivo comprimido tiene que terminar en **[formato].[compresión]**. Por ejemplo: **.sql.zip**
Buscar en su ordenador: No se eli... archivo (Máximo: 2,048KB)
También puede arrastrar un archivo en cualquier página.

Conjunto de caracteres del archivo:

Importación parcial:
 Permitir la interrupción de una importación en caso que el script detecte que se ha acercado al límite de tiempo PHP. (Esto podría ser un buen método para importar archivos grandes; sin embargo, puede dañar las transacciones.)
Skip this number of queries (for SQL) starting from the first one:

Otras opciones:
 Habilite la revisión de las claves foráneas

Formato:

Opciones específicas al formato:
Modalidad SQL compatible:
 No utilizar AUTO_INCREMENT con el valor 0

Capítulo 4. PHP DATA OBJECT

4.1.- Introducción

La extensión PHP Data Objects (PDO) define una interfaz ligera, para tener acceso a bases de datos en PHP. PDO proporciona una capa de abstracción de acceso a datos, que significa que, independientemente de la base de datos que está utilizando, se utiliza las mismas funciones para realizar consultas y obtener datos.

La necesidad de una librería como PDO nace de la gran cantidad de bases de datos soportadas por PHP. Cada una de estas bases de datos necesita de su propia API para hacer tareas generalmente comunes al resto. Esto provoca que nuestro código no sea fácil portar de una base de datos a otra y ayuda (con la colaboración de muchas otras cosas) a complicarnos la vida.

No hay que olvidar que PDO está completamente orientado a objetos, lo que facilita su uso frente al engorro de las decenas de diferentes funciones de las API de cada sistema de bases de datos.

PDO actualmente soporta los siguientes drivers para manejadores de bases de datos:

| SUPPORTED DRIVERS FOR DATABASE | |
|--------------------------------|--|
| Driver name | Supported databases |
| PDO DBLIB | FreeTDS / Microsoft SQL Server / Sybase |
| PDO FIREBIRD | Firebird/Interbase 6 |
| PDO IBM | IBM DB2 |
| PDO INFORMIX | IBM Informix Dynamic Server |
| PDO MYSQL | MySQL 3.x/4.x/5.x |
| PDO OCI | Oracle Call Interface |
| PDO ODBC | ODBC v3 (IBM DB2, unixODBC and win32 ODBC) |
| PDO PGSQ L | PostgreSQL |
| PDO SQLITE | SQLite 3 and SQLite 2 |

La instalación de PDO requiere que se edite el archivo php.ini de forma tal que la extensión PDO sea cargada automáticamente cuando se ejecuta PHP. Además, se debe habilitar el driver específico en el mismo archivo. Debe asegurarse que la línea de la extensión del driver en particular esté en el archivo y que el archivo .dll o .so esté en la carpeta de las extensiones. La ruta de las extensiones también se encuentra en el php.ini en la sección extension_dir:

```
extension=php_pdo.dll
extension=php_pdo_firebird.dll
extension=php_pdo_informix.dll
extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
extension=php_pdo_oci.dll
extension=php_pdo_oci8.dll
extension=php_pdo_odbc.dll
extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
```

4.2.- Conexión a Base de Datos

Para conectar la aplicación a la base de datos se debe instanciar la clase PDO. El constructor de la clase PDO recibe 3 parámetros:

- dsn: donde se indican los parámetros de la conexión.
- usuario: para conectarnos al servidor de base de datos.
- password: la contraseña del usuario.

En forma general, se debe ejecutar la instrucción:

```
$conexion=new Pdo( $dsn, $usuario, $password)
```

El DSN debe tener la siguiente estructura:

```
"tipodb: host=ip ; dbname=nombredbd"
```

Donde "tipobd" es el manejador de base de datos al cual se va a conectar (mysql, postgre, etc), "host" es la dirección ip donde está corriendo el servidor de base de datos (localhost si está corriendo en la misma PC) y "nombrebd" es el nombre de la base de datos a la que intenta conectarse. Por ejemplo:

```
$dsn="mysql: host=localhost; dbname=prueba";
```

Con el dsn del ejemplo anterior el script se intentará conectar a un servidor Mysql que se está ejecutando localmente y a una base de datos llamada "prueba".

Por ejemplo:



La clase Pdo contiene los siguientes métodos:

ACADEMIA DE SOFTWARE

- beginTransaction: inicio de transacciones (se verá más adelante).
- commit: completar transacción (se verá más adelante).
- errorInfo: información de error de la última operación.
- errorCode: código de error de la última operación.
- exec: ejecutar instrucciones de acción (se verá más adelante).
- prepare: instrucciones preparadas (se verá más adelante).
- rollBack: cancelar una transacción (se verá más adelante).
- lastInsertId: id del último insert (se verá más adelante).
- inTransaction: si está en medio de una transacción.

4.3.- Errores de Conexión

Si ocurre algún error en la conexión, se dispara una excepción del tipo PDOException. Se puede capturar la excepción si se necesita manejar el error. Si el script no captura el error

del constructor PDO, la acción por defecto del motor zend es terminar el script y mostrar el trazado de ejecución. Si la conexión es exitosa se retorna una instancia a un objeto PDO y la conexión permanece activa mientras viva el objeto. Para cerrar la conexión debe destruir el objeto asignándole un valor null. Si no se hace esto explícitamente, PHP automáticamente cerrará la conexión al terminar el script.

El siguiente es un ejemplo de una conexión a una base de datos MySQL con manejo del error de conexión:

```
<?php
try
{
    $dsn="mysql:host=localhost;dbname=prueba";
    $conexion=new PDO($dsn,"root","mysql");
    echo "se conecto exitosamente";
} catch(Exception $e)
{
    echo "no se pudo conectar a la base de datos. Error: ".$e->getMessage();
}
?>
```

ACADEMIA DE SOFTWARE

Capítulo 5. CONSULTAR INFORMACIÓN

5.1.- Ejecutar Consultas

Para ver los registros contenidos en una tabla se debe ejecutar una instrucción SQL del tipo "select", donde se especifica la tabla y los campos que se desean consultar. Opcionalmente también se puede especificar un filtro (usando el where en la instrucción select). Para consultar una base de datos se usa el método "query" de la clase PDO luego de instanciarse. El método "query" recibe por parámetro la instrucción SQL. El siguiente es un ejemplo de uso del método "query":

```
2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from cliente");
8
9  }catch(PDOException $e)
10 {
11     echo "Error de conexión ".$e->getMessage();
12 }
13
14 ?>
```

Al ejecutar una instrucción SQL del tipo select, el método "query" devolverá una instancia a un objeto de la clase "PDOStatement ", que permitirá manipular los registros que retorna la consulta. Algunos métodos de esta clase son los siguientes:

- columnCount: cantidad de columnas resultantes.
- fetch: retorna el próximo registro resultante.
- fetchAll: retorna todos los registros.
- fetchObject: retorna la siguiente fila como un objeto.
- rowCount: cantidad de registros que retorna la consulta.
- errorCode: código de error de la última operación.

- `errorInfo`: obtiene información extendida del error asociado con la última operación del manejador de la base de datos.

Un ejemplo de como usar métodos de la clase PDOStatement es el siguiente:

```

2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from cliente");
8      echo "El numero de registros es: ".$resultado->rowCount();
9
10 }catch(PDOException $e)
11 {
12     echo "Error de conexion ".$e->getMessage();
13 }
14

```

5.2.- Acceder a Los Registros

Luego de ejecutar la consulta se debe acceder a los registros. Para esto, se pueden usar los métodos "fetch", "fetchAll" o "fetchObject". La diferencia entre estos métodos es lo que retornan: un arreglo, arreglo de arreglos o un objeto. El método "fetch" retorna un registro o false si no hay registros. El siguiente ejemplo muestra como usar el método "fetch" para obtener el primer registro resultante de la consulta:

```

2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from cliente");
8      $registro = $resultado->fetch();
9
10 }catch(PDOException $e)
11 {
12     echo "Error de conexion ".$e->getMessage();
13 }
14 ?>

```

Si se necesita recorrer todos los registros resultantes de la consulta, se debe hacer un ciclo (while o foreach). La función "fetch" retorna false cuando llega al final (después del último registro). La forma general es la siguiente:

```

2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from talumno");
8      while ($registro = $resultado->fetch())
9      {
10          var_dump($registro);
11      }
12  }
13 catch(PDOException $e)
14 {
15     echo "Error de conexion ".$e->getMessage();
16 }
```

También se puede usar un ciclo "foreach" que permitirá recorrer todos los registros resultantes de la consulta:

ACADEMIA DE SOFTWARE

```

2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from talumno");
8      foreach ($resultado as $registro)
9      {
10          var_dump($registro);
11      }
12  }
13 catch(PDOException $e)
14 {
15     echo "Error de conexion ".$e->getMessage();
16 }
17 ?>
```

5.3.- Acceder a Los Campos

Al tomarse los datos del registro actual, se necesita acceder a los valores de los campos. La forma de acceder a los campos dependerá del FetchMode (modo de recuperación) establecido. Por defecto, el modo de recuperación es PDO::FETCH_BOTH, pero este puede cambiarse con el método "setFetchMode". Existe varios modos de recuperación:

- PDO::FETCH_ASSOC: debe devolver cada fila como un array indexado por los nombres de las columnas.
- PDO::FETCH_NAMED: igual que el anterior, la diferencia entre uno y otro se da cuando en la consulta hay varios campos con el mismo nombre. En el modo anterior, se devuelve un único valor, en este modo se devuelve un arreglo.
- PDO::FETCH_NUM: debe devolver cada fila como un array indexado por los números de columna devueltos.
- PDO::FETCH_BOTH: debe devolver cada fila como un array indexado tanto por los nombres como por los números de las columnas.

Otros modos de recuperación son los siguientes:

- PDO::FETCH_OBJ: debe devolver cada fila como un objeto con los nombres de sus propiedades equivalentes a los nombres de las columnas devueltos.
- PDO::FETCH_COLUMN: debe devolver una única columna solicitada de la siguiente fila del conjunto de resultados.
- PDO::FETCH_CLASS: debe devolver una nueva instancia de la clase solicitada, haciendo corresponder las columnas con los nombres de las propiedades de la clase.
- PDO::FETCH_INTO: debe actualizar una instancia existente de la clase solicitada, haciendo corresponder las columnas con los nombres de las propiedades de la clase.

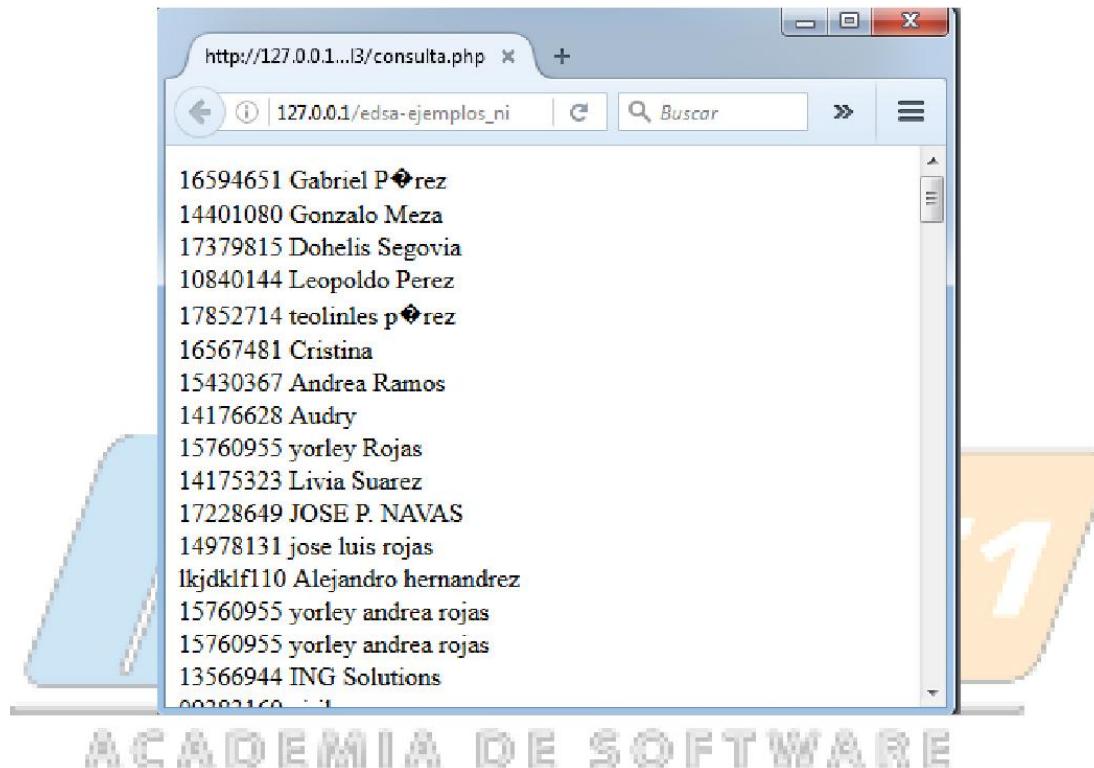
Por ejemplo, si se tiene una tabla llamada "alumno" con los campos: cedula y nombre el ejemplo para acceder a los campos sería el siguiente:

```
2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from cliente");
8      foreach ($resultado as $registro)
9      {
10          echo $registro["cedula"]." ";
11          echo $registro["nombre"]."<br>";
12      }
13  }
14 }catch(PDOException $e)
15 {
16     echo "Error de conexion ".$e->getMessage();
17 }
18 ?>
```

Si se necesita acceder a los campos de la consulta como si fueran los atributos de un objeto se establece "fetchMode" a "PDO::FETCH_OBJ" ejecutando el método setFetchMode(PDO::FETCH_OBJ) antes de recorrer el resultado de la consulta. El siguiente es un ejemplo:

```
2  <?php
3  $dsn="mysql:host=localhost;dbname=bdcadi";
4  try
5  {
6      $conexion = new Pdo($dsn,"root","mysql");
7      $resultado=$conexion->query("select * from talumno");
8      $resultado->setFetchMode(PDO::FETCH_OBJ);
9      foreach ($resultado as $registro)
10     {
11         echo $registro->cedula." ";
12         echo $registro->nombre."<br>";
13     }
14 }
15 }catch(PDOException $e)
16 {
17     echo "Error de conexion ".$e->getMessage();
18 }
19 ?>
```

La página resultante mostraría algo como esto:



ACADEMIA DE SOFTWARE

Capítulo 6. BUSCAR REGISTROS

6.1.- Formulario

Para buscar en una base de datos MySQL se necesita una página con un formulario y un cuadro de texto para que el usuario indique el registro que quiere buscar. La búsqueda se realiza ejecutando una instrucción "select" usando la cláusula "where". Un formulario sencillo para buscar un cliente por cedula tendría un aspecto como el siguiente:

```
23  <!DOCTYPE html>
24  <html>
25  <head>
26      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
27      <title></title>
28  </head>
29  <body>
30      <form name="form1" method="POST" action="buscar.php">
31          Cedula
32          <input type="text" name="txtcedula" value="" />
33          <input type="submit" value="Buscar" name="btnbuscar" />
34      </form>
35  </body>
36 </html>
```

6.2.- Procesar la Búsqueda

El formulario le envía los datos a la página `buscar.php`, la cual contiene el siguiente código:

```
5  $conexion = new Pdo($dsn,"root","mysql");
6  $buscado=$_POST['txtcedula'];
7  $resultado=$conexion->query("select * from cliente ".
8                                "where cedula='".$buscado"'");
9  $resultado->setFetchMode(PDO::FETCH_OBJ);
10 if ($resultado->rowCount()==0)
11     echo "No existe el cliente buscado";
12 else
13 {
14     $registro = $resultado->fetch();
15     echo "El cliente se llama ".$registro->nombre;
16 }
```

Si necesita buscar varios registros que cumplan con la condición, se puede utilizar el operador "like" con el comodín "%". El siguiente es un ejemplo:

```
5 $conexion = new Pdo($dsn,"root","mysql");
6 $buscado=$_POST['txtcedula'];
7 $resultado=$conexion->query("select * from cliente ".
8                                     "where cedula like '$buscado%'");
9 $resultado->setFetchMode(PDO::FETCH_OBJ);
10 if ($resultado->rowCount() == 0)
11     echo "No existe el cliente buscado";
12 else
13 {
14     echo "Registros encontrados:";
15     foreach ($resultado as $registro)
16     {
17         echo $registro->cedula." ";
18         echo $registro->nombre."<br>";
19     }
20 }
```



ACADEMIA DE SOFTWARE

Capítulo 7. LISTADOS

7.1.- Llenar Select

Una de las tareas más comunes en una página dinámica es mostrar al usuario una lista desplegable con los valores que están contenidos en una tabla, por ejemplo: listado de modelos de carros, listado de contactos, de ciudades, etc. Es básicamente el mismo procedimiento que cuando se llena una tabla, pero adaptado al código HTML de los select. El siguiente código es un ejemplo:

```
1  <?php
2  $dsn="mysql:host=localhost;dbname=bdcadi";
3  try
4  {
5      $conexion = new PDO($dsn,"root","mysql");
6      $resultado=$conexion->query("select * from ciudad");
7      $resultado->setFetchMode(PDO::FETCH_OBJ);
8  }catch(PDOException $e)
9  {
10     echo "Error de conexion ".$e->getMessage();
11 }
12 ?>
13 <!DOCTYPE html>
14 <html>
15 <head>
16     <title></title>
17 </head>
18 <body>
19 <select name="ciudad">
20 <?php
21     foreach ($resultado as $registro)
22         echo "<option>$registro->nombre</option>";
23 <?>
24     </select>
25 </body>
26 </html>
```

El código generado será el siguiente:

Vista en el navegador

Ciudades cabudare

Código HTML resultante

```

1 Ciudades
2 <select name="txtcuidad">
3   <option>cabudare</option>
4   <option>barquisimeto</option>
5   <option>caracas</option>
6   <option>Valencia</option>
7   <option>Merida</option>
8 </select>
9
10

```

7.2.- La Propiedad Value

Cuando se envía la información de un select, se envía lo que está seleccionado. En ocasiones se necesita enviar un código asociado al valor seleccionado, que sirva como clave primaria o identificador único del elemento. Esto mayormente es necesario cuando se trabajan con bases de datos relacionales, donde lo más común es trabajar con claves foráneas. Es este caso se debe agregar al ejemplo anterior la propiedad "value":

```

<select name="ciudad">
<?php
    foreach ($resultado as $registro)
        echo "<option value='".$registro->codigo'>$registro->nombre</option>";
    ?>
</select>

```

El código HTML generado es el siguiente:

Vista en el navegador

Ciudades cabudare

Código HTML resultante

```

1 Ciudades
2 <select name="txtcuidad">
3   <option value=1>cabudare</option>
4   <option value=2>barquisimeto</option>
5   <option value=3>caracas</option>
6   <option value=4>Valencia</option>
7   <option value=5>Merida</option>
8 </select>
9
10

```

Cuando una página se carga, el navegador por defecto coloca como seleccionado el primer elemento de un select. Para hacer que quede seleccionado algún elemento en particular, se debe usar la propiedad "selected" de la etiqueta "option". Este valor "selected" deberá ser colocado dinámicamente al elemento que se desea aparezca seleccionado.

```
<?php
    if (isset($_POST['ciudad']))
        $seleccionado=$_POST['ciudad'];
    else
        $seleccionado=0;
?>
<form name="form1" method="post">
    <select name="ciudad">
        <?php
            foreach ($resultado as $registro)
            {
                if ($registro->codigo==$seleccionado)
                    $x="selected";
                else
                    $x="";
                echo "<option $x value='".$registro->codigo'>
                    $registro->nombre
                </option>";
            }
        ?>
        </select>
    </form>
```

Se puede crear en una librería una función que permita la creación de un select dada una tabla, para reutilizarla y así simplificar esta tarea:

```
13     function crear_select($nombre,$tabla,$campo,$id,$seleccionado)
14     {
15         $resultado=$conexion->query("select * from $tabla");
16         echo "<select name='".$nombre."'";
17         foreach ($resultado as $registro)
18         {
19             if ($registro[$id]==$seleccionado)
20                 $x="selected";
21             else
22                 $x="";
23             echo "<option $x value='".$registro[$id]."'".
24                 $registro[$campo].
25                 "</option>";
26         }
27         echo "</select>";
28     }
```

El código para llamar la función es el siguiente:

```
<?php
    if (isset($_POST['ciudad']))
        $seleccionado=$_POST['ciudad'];
    else
        $seleccionado=0;
?>
<form name="form1" method="post">
    <?php
        crear_select("ciudad","ciudad","nombre","id",$seleccionado);
    ?>
</form>
```

Capítulo 8. INCLUIR REGISTROS

8.1.- Incluir Registros

Para incluir un registro en una tabla de MySQL necesitamos una página que contenga un formulario HTML donde se soliciten los datos al usuario. Continuando con el ejemplo de la tabla "clientes" el formulario tendría un diseño como el siguiente:

```
28 |<html>
29 |  <head>
30 |    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
31 |    <title>Registro de clientes</title>
32 |  </head>
33 |  <body>
34 |    <h1>Registro de Clientes</h1>
35 |    <form name="formulario" method="POST">
36 |        Cedula
37 |        <input type="text" name="txtcedula" value="" />
38 |        Nombre
39 |        <input type="text" name="txtnombre" value="" />
40 |        Saldo
41 |        <input type="text" name="txtsaldo" value="" />
42 |        <input type="submit" value="Guardar" name="guardar" />
43 |    </form>
44 |  </body>
45 |</html>
```

ACADEMIA DE SOFTWARE

Para ejecutar las acciones vamos a utilizar el método exec(instrucción), pasándole por parámetro la instrucción sql : insert, update o delete que se necesita ejecutar. Para incluir, la página que recibe los datos de este formulario debe ejecutar una instrucción SQL del tipo "insert" donde se utilicen los datos captados en el formulario. La página que procesa el formulario tendrá algo así:

```
1  <?php
2  if (isset($_POST["guardar"]))
3  {
4      $dsn="mysql:host=localhost;dbname=bdcadi";
5      try
6      {
7          $conexion = new Pdo($dsn,"root","mysql");
8          $c=$_POST['txtcedula'];
9          $n=$_POST['txtnombre'];
10         $s=$_POST['txsaldo'];
11         $sql="insert into cliente (cedula,nombre,saldo)
12             values
13             ('$c','$n','$s')";
14         $conexion->exec($sql);
15         echo "Registro exitoso ";
16     }
17 } catch(PDOException $e)
18 {
19     echo "Error de conexion ".$e->getMessage();
20 }
21 }
22 ?>
```

Se puede determinar si el registro fue efectivamente insertado evaluando el valor que retorna el método "exec". Si retorna 0 indica que no logró insertar el registro. Por ejemplo:

```
$sql="insert into cliente (cedula,nombre,saldo)
      values
      ('$c','$n','$s')";
$resultado=$conexion->exec($sql);
if ($resultado==0)
    echo "Ocurrio un error al registrar";
else
    echo "Registro exitoso ";
```

8.2.- Validación

Antes de incluir los registros puede ser necesario validar los datos. La validación puede ser muy particular para cada caso, aunque la mínima validación podría ser evitar que los datos estén vacíos (asumiendo que existen, es decir, que se verificó la existencia de los datos con la función isset). Por ejemplo, de validación:

```
7 | $conexion = new Pdo($dsn,"root","mysql");
8 | if (empty($_POST['txtcedula']))
9 |     echo "La cédula no puede estar vacío";
10| else
11|     if (empty($_POST['txtnomre']))
12|         echo "El nombre no puede estar vacío";
13|     else
14|     {
15|         $c=$_POST['txtcedula'];
16|         $n=$_POST['txtnombre'];
17|         $s=$_POST['txsaldo'];
18|         $sql="insert into cliente (cedula,nombre,saldo)
19|             values
20|             ('$c','$n','$s')";
21|         $resultado=$conexion->exec($sql);
22|         if ($resultado==0)
23|             echo "Ocurrio un error al registrar";
24|         else
25|             echo "Registro exitoso ";
26|     }
```

Capítulo 9. MODIFICAR REGISTROS

9.1.- Modificar un Registro

Para modificar un registro en una tabla de MySQL se necesita una página que contenga un formulario HTML donde se soliciten los datos al usuario. Continuando con el ejemplo de la tabla "clientes" el formulario tendría un diseño como el siguiente:

```
28 |<html>
29 |  <head>
30 |    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
31 |    <title>Registro de clientes</title>
32 |  </head>
33 |  <body>
34 |    <h1>Registro de Clientes</h1>
35 |    <form name="form1" method="POST">
36 |        Cedula
37 |        <input type="text" name="txtcedula" value="" />
38 |        Nombre
39 |        <input type="text" name="txtnombre" value="" />
40 |        Saldo
41 |        <input type="text" name="txtsaldo" value="" />
42 |        <input type="submit" value="Guardar" name="guardar" />
43 |    </form>
44 |  </body>
45 |</html>
```

ACADEMIA DE SOFTWARE

La página que recibe los datos de este formulario debe ejecutar una instrucción SQL del tipo "update" donde se utilicen los datos recibidos por el formulario. La página que procesa el formulario tendrá algo así:

```
1  <?php
2  if (isset($_POST["guardar"]))
3  {
4      $dsn="mysql:host=localhost;dbname=bdcadi";
5      try
6      {
7          $conexion = new Pdo($dsn,"root","mysql");
8
9          $c=$_POST['txtcedula'];
10         $n=$_POST['txtnombre'];
11         $s=$_POST['txsaldo'];
12         $sql="update cliente set nombre='$n', saldo=$s
13             where cedula='$c'";
14         $resultado=$conexion->exec($sql);
15         if ($resultado==0)
16             echo "Ocurrio un error al actualizar";
17         else
18             echo "Actualizacion exitosa ";
19
20     }catch(PDOException $e)
21     {
22         echo "Error de conexion ".$e->getMessage();
23     }
24 }
25 ?>
```

Capítulo 10. ELIMINAR

10.1.- Eliminar un Registro

La eliminación tiende a ser complicada debido a la interfaz de usuario para que se utilice. Una manera es solicitando al usuario un valor que identifique a un registro como único (que corresponda con la clave primaria). Una búsqueda con un formulario como el visto en el tema de buscar podría ser una opción. Luego la idea es ejecutar una instrucción "delete" de SQL usando el método "exec" de Pdo:

```
<?php
1   $dsn="mysql:host=localhost;dbname=bdcadi";
2   try
3   {
4       $conexion = new Pdo($dsn,"root","mysql");
5       $buscado=$_POST['txtcedula'];
6       $resultado=$conexion->exec("delete from cliente ".
7                                     "where cedula='".$buscado."'");
8       if ($resultado->rowCount()==0)
9           echo "No se pudo eliminar el cliente $buscado";
10      else
11          echo "Se elimino exitosamente el cliente";
12    }catch(PDOException $e)
13    {
14        echo "Error de conexion ".$e->getMessage();
15    }
16
17
18 ?>
```

Capítulo 11. ELIMINAR VARIOS REGISTROS

11.1.- Introducción

Otra forma de eliminar es colocando un listado de todos los registros, colocar al lado de cada registro un checkbox, de forma tal el usuario pueda seleccionar varios registros y luego hacer click en un botón eliminar (estilo webmail). Esta forma de hacerlo es más amigable para el usuario, pero más complicada de programar.



| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Asunto | Fecha | Tamaño |
|--------------------------|-------------------------------|---|------------------|--------|
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Recomendaciones de seguridad para los usuarios de Cantv.net | 22/01/2009 15:33 | 2Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Atrapa corazones viaja a Los Roques con tu pareja | 10/02/2009 08:57 | 2Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Sistema Automatizado CADIVI - Enviado 24.08.2009 - 20:54:20 | 25/08/2009 09:41 | 4Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Fw: NUEVA MERCANCIA COMERCIAL DT 17 | 25/08/2009 18:15 | 1585Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Fw: NUEVA LISTA COMERCIAL DT 17 CON IMAGENES DE SILLAS Y MESAS... | 27/08/2009 10:25 | 828Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Alva Fuentes | 28/08/2009 02:50 | 6Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | Fw: ANEXO MERCANCIA DISPONIBLE PARA LUNES 07/09 COMERCIAL DT. | 04/09/2009 09:54 | 30Kb |
| <input type="checkbox"/> | <input type="checkbox"/> 0 De | MySQL Newsletter: September 2009 | 16/09/2009 21:50 | 47Kb |

11.2.- Formulario

Lo primero que se debe hacer es agregar una etiqueta de formulario, una tabla con 2 filas y por lo menos 2 columnas (opcional para organizar). Luego, agregar un checkbox en la primera columna, la información que se identifica a cada registro visualmente en la segunda columna, y al final, un botón de tipo enviar con el valor "Eliminar", como se muestra en la imagen:

```

<?php

$resultado=$conexion->query("select * from cliente");
?>
<form name="form1" method="POST">
    <table border="1" width="250">
        <tr><td></td><td>Nombre</td></tr> Se llenan las filas de la tabla con los registros
        <?php
        foreach ($resultado as $registro)
        {
        ?>
            <tr>
                <td><input type="checkbox" name="marcados[]" value="" /></td>
                <td><?php echo $registro['nombre']?></td>
            </tr>
        <?php
        }
        ?>

    </table>
    <input type="submit" value="Eliminar" name="bteliminar" />
</form>

```

Se debe tener la precaución de colocarle al checkbox un nombre cualquiera, pero colocándole al final del nombre unos corchetes vacíos [] y al valor del checkbox debemos imprimirle un campo que sea clave primaria en la tabla de la base de datos que vamos a listar. En el siguiente ejemplo se muestra un listado de clientes y se usa como clave primaria la cédula:

```

<?php
foreach ($resultado as $registro)
{
?>
    <tr>
        <td><input type="checkbox" name="marcados[]" value="<?php echo $registro['cedula']?>" /></td>
        <td><?php echo $registro['nombre']?></td>
    </tr>
<?php
}
?>

```

El nombre lleva corchetes[]

Valor con un campo que sea clave primaria en la tabla de la base de datos

El resultado en el navegador es el siguiente:

| Nombre | |
|--------------------------|----------------|
| <input type="checkbox"/> | jose rojas |
| <input type="checkbox"/> | yure morillo |
| <input type="checkbox"/> | maria camacaro |
| <input type="checkbox"/> | blanca dellan |
| Eliminar | |

El código HTML generado es el siguiente:

```
<table border="1" width="250">
<tr>
    <td></td> <td>Nombre</td>
</tr>
<tr>
    <td><input type="checkbox" name="marcados[]" value="14978131" /></td>
    <td>jose rojas</td>
</tr>
<tr>
    <td><input type="checkbox" name="marcados[]" value="14978412" /></td>
    <td>yure morillo</td>
</tr>
<tr>
    <td><input type="checkbox" name="marcados[]" value="4045640" /></td>
    <td>blanca dellan</td>
</tr>
</table>
```

11.3.- Procesar Los Elementos

Luego se debe colocar el código PHP para eliminar los elementos marcados en los checkboxes. Debe colocarse antes del código que genera el listado que se va a mostrar para dar el efecto de eliminación al recargarse la página:

```
<?php  
$dsn="mysql:host=localhost;dbname=bdcadi";  
$conexion = new Pdo($dsn,"root","mysql");  
if (isset($_POST['marcados']))  
{  
    $marcados=$_POST['marcados'];  
    for ($i=0;$i<count($marcados);$i++)  
        $conexion->exec("delete from cliente ".  
                           "where cedula='".$marcados[$i']."'");  
}  
$resultado=$conexion->query("select * from cliente");  
?>
```



ACADEMIA DE SOFTWARE

Capítulo 12. INSTRUCCIONES PREPARADAS

12.1.- Instrucciones Preparadas

En ocasiones vamos a necesitar ejecutar varias veces la misma instrucción SQL (de consulta o de acción) pero con distintos valores. Para estos casos podemos utilizar instrucciones preparadas. Las instrucciones preparadas ofrecen 2 beneficios mayores:

- Las consultas sólo necesitan ser parseadas una sola vez, pero pueden ser ejecutadas varias veces con diferentes parámetros. Cuando la consulta es preparada, la base de datos será analizada, la consulta se compila y se optimiza para ser ejecutada. Para consultas complejas este proceso puede tomar un tiempo tal que puede hacer que la aplicación tenga un desempeño menor si es necesario repetir varias veces la misma consulta con parámetros diferentes. Usando instrucciones preparadas, la aplicación se evita de repetir el proceso analizar/compilar/optimizar. Por consecuencia, las instrucciones preparadas usan menos recursos y se ejecutan más rápido.
- Las instrucciones preparadas no necesitan aplicársele encomillado a los datos, debido a que el driver se encarga automáticamente de manejar esta tarea. Si una aplicación usa exclusivamente instrucciones preparadas, el programador puede estar seguro de que no ocurrirá inyección de SQL.

Las instrucciones preparadas son muy útiles debido a que PDO puede emular esta funcionalidad, aunque el manejador de base de datos no soporte esta función. Esto asegura que una aplicación estará habilitada para usar este paradigma sin importar las capacidades del manejador. Para preparar las instrucciones SQL usaremos el método "prepare" de la clase PDO, y luego para ejecutarlas usamos el método "execute".

12.2.- Parámetros en Instrucciones Preparadas

El uso principal de las instrucciones preparadas es el de tener instrucciones SQL con parámetros, que luego serán sustituidos por valores en tiempo de ejecución. Veamos un ejemplo:

```
<?php
try
{
    $dsn="mysql:host=localhost;dbname=prueba";
    $conexion=new PDO($dsn,"root","mysql");
    echo "se conecto exitosamente";
}catch(Exception $e)
{
    echo "no se pudo conectar a la base de datos. Error: ".$e->getMessage();
}

$stmt = $conexion->prepare("INSERT INTO cliente (CEDULA, NOMBRE) VALUES (:ced, :nom)");
$stmt->bindParam(':ced', $cedula);
$stmt->bindParam(':nom', $nombre);

// insert one row
$nombre = 'jose';
$cedula = '12345';
$stmt->execute();

// insert another row with different values
$nombre = 'luis';
$cedula = '264545';
$stmt->execute();
```

