

title: Clase 3 Proyecto 2016
Author: Einar Lanfranco, Claudia Banchoff
description: Acceso a BBDD usando PHP + Modelo MVC + Objetos en PHP
keywords: base de datos
css: proyecto.css

Proyecto de Software

Cursada 2016

Hoy seguimos con ...



Temario

- Repaso Clase Anterior
 - **PHP**
 - BBDD
 - Objetos
-

Cookies

- Básicamente, son “tokens” en el requerimiento HTTP que permite identificar de alguna manera al cliente en el servidor.
 - Se almacenan en el disco del cliente.
 - Muy usado por ser HTTP un protocolo sin estado
 - Formato: **nombreCookie=valor;expires=fecha;**
 - PHP las considera variables externas: Usa `$_COOKIE` (un arreglo con las cookies generadas)
 - Mediante la función `setcookie()` es posible grabar cookies en el cliente.
 - Veamos un ejemplo de uso de [cookies](#)
-

class: destacado

Sesiones

Es un mecanismo para conservar ciertos datos a lo largo de varios accesos.

- Permite registrar un número arbitrario de variables que se conservarán en las siguientes peticiones.
 - Identificador: A cada visitante se le asigna un identificador único, llamado **session id** (identificador de sesión).
 - **Hay dos formas de propagar un identificador de sesión:**
 - Mediante cookies
 - A través de la URL.
-

Sesiones

- **Algunas funciones:**
 - **session_start():** Inicializa los datos de una sesión.
 - **session_destroy():** Destruye los datos guardados en una sesión.
 - **session_id():** recupera o modifica el valor de la sesión.
 - **Debe ubicarse antes que cualquier salida al browser.**
 - Si no existe un **session ID**, `session_start()` lo crea e inicializa un arreglo vacío denominado `$_SESSION`.
-

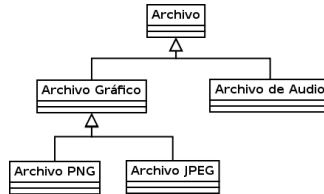
Sesiones

- Los archivos con los datos de la sesión se generan en el servidor, usualmente en el directorio tmp del servidor.
- El cliente sólo guarda el session Id

Veamos el ejemplo de [sesiones](#)

POO en PHP

- PHP tiene soporte para construir clases.
- Es posible definir clases en una jerarquía con herencia simple.
- Palabra reservada **class**



Clases

- Clase Archivo:

```
<?php
class Archivo{
    // variables de instancia
    // métodos
}
```

- Clase Archivo gráfico que hereda de Archivo:


```
<?php
class ArchivoGrafico extends Archivo{
}
```

Constructores

- **Constructores:** En PHP 5 estas funciones se llaman `__construct()`. Aunque por compatibilidad con versiones anteriores, si no se encuentra esta función se busca una cuyo nombre sea igual al de la clase.
-

Constructores

```
class Empleado {
    .....
    function __construct($s=100) { $this->basico=$s; }
}
class EmpleadoContratado extends Empleado{
    .....
    function __construct(){
        parent::__construct(20);.....
    }
}
```



El constructor de un objeto NO invoca automáticamente al constructor de la clase base.

Destruyores:

- Cuando se destruye la última referencia a un objeto, se invoca el método “destructor” de ese objeto.
- Este método se llama **__destruct()** y no recibe parámetros.
- Al igual que en el caso de los constructores, los métodos destructores de un objeto NO invocan automáticamente los destructores de las clase base. Deben invocarse explícitamente: **parent::__destruct()** en el cuerpo del método.

Instancias de una clase

Operador new:

\$x = new nombreDeLaClase();

```
<?php
class Persona {
    protected $nombre;
    public $dni;
    function verNombre(){
        return $this->nombre;
    }
}
$per1=new Persona();
$per2=new Persona();
$per1->verNombre();
$per2->dni=$per1->dni;
?>
```

Ejemplo de clase en PHP

Ver [clases](#)

Visibilidad de las propiedades

- **Público:** Los miembros de clases declarados como **public** pueden ser accedidos desde cualquier lado
 - **Privado:** Aquellos miembros definidos como **private**, únicamente pueden ser accedidos desde la clase que los definió.
 - **Protegido:** Los miembros declarados como **protected**, sólo desde la misma clase, desde las clases que hereden de ella y desde las clases parent.
-

Ejemplo:

```
<?php
class ClaseA {
    public $a1;
    protected $a2;
    private $a3;
    .....
}
class ClaseB extends ClaseA {
    public $b1;
    .....
}
```

```
$objA= new ClaseA;
$objB= new ClaseB;

print $objA->a1; // OK
print $objA->a2; // Error
print $objA->a3; // Error

¿print $objB->a1;✓
¿print $objB->b1;✓
¿print $objB->a2;✗
¿print $objB->a3;✗ ➡ Ni siquiera está definida
```

- <http://php.net/manual/es/language.oop5.visibility.php>
-

Varios

- Operador **instanceof**

```
<?php
if ( $emple instanceof Empleado ) {
    echo "Esto es un empleado";
}

?>
```

(::) Resolución de Alcance

Algunas veces es útil hacer referencia a funciones y variables en las clases base o hacer referencia a funciones que se encuentran en clases que aún no tienen ninguna instancia. El operador **::** es utilizado para realizar esto.

```
<?php
class MyClass {
    const CONST_VALUE = 'Un valor constante';
}
echo MyClass::CONST_VALUE;
?>
```

```
include("MDB2.php")
$conDB =
MDB2::connect("mysql://usuario:clave@host/BaseDeDatos");
```

Enlace estático en tiempo de ejecución

- Permite hacer referencias a la clase en uso dentro de un contexto de herencia estática
 - Enlaces estáticos **parent**, **self** y **static**. Ver [Ejemplo](#)
 - Referencia: <http://php.net/manual/es/language.oop5.late-static-bindings.php>
-

Miembros static

- Una propiedad declarada como static no puede ser accedida con un objeto de la clase. Sí a través de métodos estáticos.

```
class Libro {  
    public static $genCodigos = 0;  
    private $codigo;  
  
    function __construct(){  
        $this->codigo=self::$genCodigos++;  
    }  
    function verCodigo(){  
        return $this->codigo;  
    }  
}
```

```
print Libro::$genCodigos; ✓  
$unLibro = new Libro(); ✓  
print $unLibro->genCodigos; ✗ No definido
```

Ejemplo de uso static en propiedades

- Ver [fuente](#)
-

Métodos abstractos

- Un método es abstracto si sólo declara el prototipo del método y no su implementación.
- Cuando se hereda de una clase abstracta, todos los métodos definidos como abstractos en la declaración de la clase madre deben ser definidos en la clase hija, y con una visibilidad igual o menor.

Clases Abstractas

- No tienen instancias.
 - Es cualquier clase que contenga aunque sea un método abstracto.
-

Abstractos e implementaciones

```
abstract class Figura {  
    abstract protected function calcularArea();  
    public function imprimirArea() {  
        print $this->calcularArea();  
    }  
}  
class Cuadrado extends Figura {  
    protected $lado=4;  
    public function calcularArea() {  
        return $this->lado*$this->lado; } }  
class Triangulo extends Figura {  
    protected $radio=2;  
    public function calcularArea() {  
        return 3.14 * $this->radio*$this->radio; } }  
?>
```

→ Método Abstracto

→ Implementado en cada clase

- Ver [Abstracciones](#)

Interfaces

- Son estructuras similares a las clases abstractas.
- Pueden contener sólo métodos abstractos, ninguna implementación.
- Suelen verse como catálogos de métodos a implementar.

```
<?php  
interface unaInterface {  
    function unMetodo( ); }  
?>
```

- Equivalente a:

```
<?php  
interface unaInterface {  
    public abstract function unMetodo( ); }  
?>
```

Interfaces (Cont.)

- Una clase puede implementar una o más interfaces.
- Debe proveer una implementación para todos los métodos definidos en la interface.

```
<?php  
class MiClase extends ClaseBase implements unaInterface {  
    //...  
}  
class MiClase implements unaInterface , otraInterface { //...  
}  
class OtraClase implements unaInterface {  
    //...  
}  
?>
```

Polimorfismo

- Distintos objetos comparten la misma interfaz y en ejecución se determina el código a ejecutar, dependiendo del contexto. No importa cómo lo implementa cada uno.

```
<?php
$objetos=array(new Clase1(), new Clase2(), new Clase3());
foreach ($objetos as $unObj){
    $unObj->metodoComun(); //redefinido para cada clase
    //...
}
?>
```

Excepciones

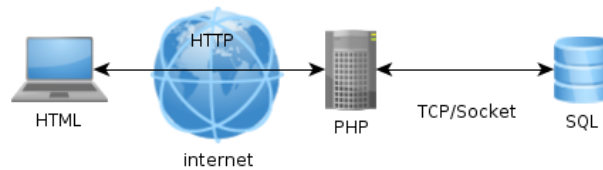
- Existe una clase **Exception**
- Es posible levantar una excepción explícitamente:
 - **throw** new Exception("my message", 44);
 - Finaliza la ejecución del método y deja disponible el objeto Exception
- Por defecto produce un error fatal
 - Para manejar las excepciones implementa un mecanismo **try/catch**
 - Cualquier código que pueda levantar una excepción debe estar encerrado dentro de una cláusula try
 - El bloque catch se utiliza para manejar la excepción

Excepciones

```
<?php
try {
    -- sentencias que pueden producir una excepción --
}
catch (Exception $e) {
    -- sentencias para manejar la excepción --
}
?>
```

Ver [excepciones](#) y [excepcionesNuevas](#)

Accediendo a Bases de Datos



Lenguaje SQL (Structured Query Language)

- Sentencias insert, update, select, etc....
- Ejemplos:
 - select * from tabla where condición
 - insert into tabla (campos) values (valores)
 - update tabla set campo1='valor1' where condición

Pasos típicos

```
<?php
//Conectarse a la BBDD
$con = conectar_a_la_BBDD("mibdd","usuario", "clave");

// Armar la consulta
$consulta = "SELECT * FROM AlgunaTabla where campo1 ='".$variable."'";

//Ejecutar la consulta
$resultado = ejecutar_consulta($consulta);

//Procesar resultados
if (tengo_resultados($resultado))
    $campo =extraigo_datos($resultado,"nombreCampo");
?>
```

Importante

MySQL: Motor de Base de Datos

SQL: Lenguaje de Consulta



Consejos

- Habilitar los logs de la manera más descriptiva posible
 - Instalar alguna aplicación que nos permita acceder de manera menos complicada a la base de datos
→ PhpMyAdmin
-

MySQL

- El archivo más importante de MySQL es su configuración que la encontramos en `/etc/mysql/my.cnf`
 - Por defecto el log esta deshabilitado por cuestiones de performance.
 - Ejemplos de configuración avanzados: `/usr/share/doc/mysql-server-5.1/examples`
-

MySQL

Habilitar el Log de todas las transacciones:

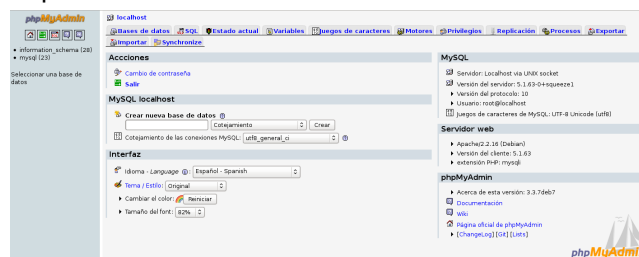
- `general_log_file = /var/log/mysql/mysql.log`
- `general_log = 1`

Logear consultas lentas y sin índices:

- `log_slow_queries = /var/log/mysql/mysql-slow.log`
 - `long_query_time = 2`
 - `log-queries-not-using-indexes`
-

PhpMyAdmin

- Interfaz de Administración de la Base de Datos MySQL
- Podemos exportar e importar a varios formatos



<http://www.phpmyadmin.net>

class: destacado

Acceso a MySQL con PHP

Esto es histórico: la extensión MySQL fue declarada obsoleta en PHP 5.5.0 y eliminada en PHP 7.0.0.

- **Tipos de recursos:** Hay dos tipos de recursos usados en el módulo MySQL.

- El identificador de enlace para una conexión de base de datos
- Un recurso que almacena el resultado de una consulta.

- **Algunas funciones....**

- `mysql_connect()`: Conecta a una base de datos
 - `mysql_select_db`: Permite seleccionar una base de datos
 - `mysql_query()`: Envía una sentencia SQL a MySQL
 - `mysql_fetch_row()`: Busca un registro en el resultado
 - `mysql_result()`: Devuelve los resultados de una consulta
-

Acceso a BBDD – MySQL

- La extensión **mysqli** y **PDO** son las opciones recomendadas actualmente que nos permite acceder a la funcionalidad provista por MySQL 4.1.2 o superior.

	ext/mysqli	PDO_MySQL	ext/mysql
Versión de PHP donde se introdujo	5.0	5.1	2.0
Incluida con PHP 5.x	SI	SI	SI
Estado de desarrollo	Activo	Activo	Sólo mantenimiento
Ciclo de vida	Activo	Activo	Obsoleto
Recomendada para nuevos proyectos	SI	SI	No
Interfaz de POO	SI	SI	No
Interfaz procedimental	SI	No	SI
La API admite la no espera, consultas asíncronas con mysqli	SI	No	No
Conexiones persistentes	SI	SI	SI
La API admite conjunto de caracteres	SI	SI	SI
La API admite sentencias preparadas del lado del servidor	SI	SI	No
La API admite sentencias preparadas del lado del cliente	No	SI	No
La API admite procedimientos almacenados	SI	SI	No
API admite sentencias múltiples	SI	La mayoría	No
La API admite transacciones	SI	SI	No
Las transacciones se pueden controlar con SQL	SI	SI	SI
Admite toda la funcionalidad de MySQL 5.1+	SI	La mayoría	No

- Ver: <http://php.net/manual/es/mysqliinfo.api.choosing.php>
-

Mysqli

- La extensión **mysqli** (mysql improved) permite acceder a la funcionalidad proporcionada por MySQL 4.1 y posterior.
 - Puede utilizarse en forma procedimental y como objetos.
 - Ver formas de utilizarlo en <http://www.php.net/manual/es/mysqli.construct.php>
-

Proceduralmente

```
<?php
//conexion:
$link = mysqli_connect($db_host,$db_user,$db_pass,$db_base) or
    die("Error " . mysqli_error($link));
//consultation:
$query = "SELECT nombre FROM usuarios" ;
//execute the query.
$result = $link->query($query) or
    die("Error en la consulta: " . mysqli_error($link));
//display information:
while($row = mysqli_fetch_array($result)) {
    echo $row["nombre"] . "<br>";
}
mysqli_close($link);
?>
```

Acceso a la Base con mysqli

Ver [accediendo](#)

Objetos

```
<?php
//conexion:
$con = new mysqli($db_host,$db_user,$db_pass);
$con->select_db($db_base);
//consultation:
$query = "SELECT nombre FROM usuarios";
//execute the query.
$result = $con->query($query);
//display information:
while($row = $result->fetch_object()) {
    echo $row->nombre . "<br>";
}
$result->close();
?>
```

Más Motores



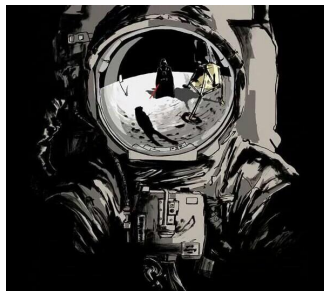
CUBRID	DB++	dBase	filePro	SQLite
Paradox	FrontBase	IBM DB2	Informix	Sybase
Ingres	MaxDB	MongoDB	mSQL	PostgreSQL
Microsoft SQL Server	MySQL	Oracle OCI8	Ovrimos SQL	Firebird/InterBase



PostgreSQL



WE HAVE A PROBLEM!!



- ¿Qué pasa si queremos migrar de motor de BDD?
- ¿Qué pasa si queremos tener múltiples BDD conectadas?

Abstrayéndonos → Otra opción PDO

- La extensión Objetos de Datos de PHP (PDO por sus siglas en inglés) define un interfaz ligera para poder acceder a bases de datos en PHP.
- Hay que usar un controlador de PDO específico para cada servidor de base de datos.

- PDO proporciona una capa de abstracción de acceso a datos, lo que significa que, independientemente de la base de datos que se esté utilizando, se usan las mismas funciones para realizar consultas y obtener datos → Permitiría cambiar de motor de base de datos.
 - Ver [drivers](#)
-

Con PDO

```
<?php
try {
    $gbd = new PDO('mysql:host=localhost;dbname=test', $usuario, $pass);
    foreach($gbd->query('SELECT * from FOO') as $fila) {
        print_r($fila);
    }
    $gbd = null;
} catch (PDOException $e) {
    print "¡Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Con PDO ejemplo 2

```
<?php
$status=$_GET["status"];
try {
    $cn = new PDO("mysql:dbname=base;host=localhost","user","pass");
    $query = $cn->prepare("SELECT * FROM usuarios WHERE status=?");
    $query->execute(array($status));
    while($row = $query->fetch())
    { echo $row['nombre'];
    }
}
catch(PDOException $e){
    echo "ERROR". $e->getMessage();
}
?>
```

Cierre de conexiones en PDO

Una vez realizada con éxito una conexión a la base de datos, será devuelta una instancia de la clase PDO al script. La conexión permanecerá activa durante el tiempo de vida del objeto PDO. Para cerrar la conexión, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas (esto se puede hacer asignando NULL a la variable que contiene el objeto). Si no se realiza explícitamente, PHP cerrará automáticamente la conexión cuando el script finalice.

Ver ejemplo con pase de parámetro

Ver

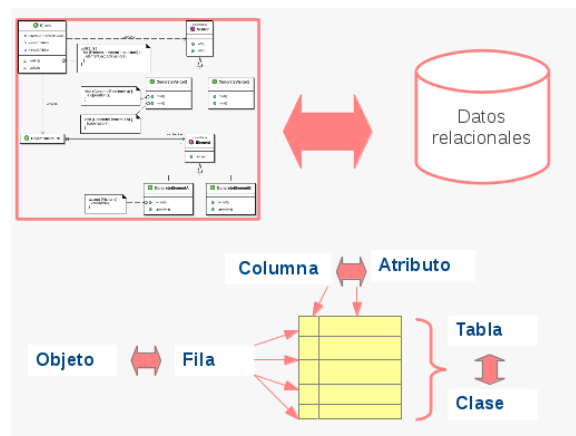
En general

- <http://php.net/manual/es/class.pdostatement.php>
 - <http://php.net/manual/es/book.mysql.php>
-

Comparando

- A mayor abstracción menos control .
 - Al abstraernos es posible que se compliquen algunas funciones propias del motor que queremos usar.
 - ¿La abstracción es necesaria?
 - Menos control con PDO que con mysql.
 - Algunas bases no están disponibles con PDO y hay que usar si o si el driver específico. Ver <http://www.php.net/manual/es/pdo.drivers.php>
-

Una vuelta de rosca más->ORM



ORM - Object-Relational Mapping

- Mapeo de Objetos a Base de Datos Relacionales.
 - Permite acceder a una base de datos relacional como si fuera orientada a objetos.
 - Transforma las llamadas a los objetos en consultas SQL, que permiten independizar el motor de BD utilizado en la aplicación.
 - De acuerdo a la implementación se utiliza una sintaxis diferente.
-

¿Por qué?

- **BBDD relacionales**
 - Datos escalares: números, cadenas de texto, etc...
 - **Aplicaciones orientadas a objetos**
 - Objetos con propiedades y métodos
 - **¿Problema?**
 - Convertir los objetos en datos escalares para grabar los datos en la Base de Datos y volver a convertirlos en objetos al recuperarlos.
-

ORM

- **Ventajas**
 - Abstracción a la BBDD
 - Reutilización de código
 - **Desventajas**
 - Otra capa intermedia
 - Otro lenguaje de consulta
 - **Algunos ORMs**
 - Propel
 - Doctrine
 - LinQ
 - Active Record
-

Referencias

- PHP -> <http://php.net>
- PhpMyAdmin -> <http://www.phpmyadmin.net>
- Lenguaje SQL -> <http://www.w3schools.com/sql/default.asp>
- MySQL -> <http://dev.mysql.com/doc/>

- PDO -> <http://www.php.net/manual/es/class.pdostatement.php>
- Mysqli -> <http://php.net/manual/es/mysqli.prepare.php>