

# Trabajar con frameworks de PHP

Desarrollo de Aplicaciones Web con  
Software Interpretado en el Servidor

# Objetivos

1. Tener claridad sobre el funcionamiento del Modelo Vista Controlador (MVC).
2. Desarrollar las habilidades para descargar e instalar CodeIgniter en un servidor web con soporte para PHP.
3. Adquirir destrezas para utilizar el framework CodeIgniter en el desarrollo de aplicaciones web del lado del servidor.
4. Lograr el dominio de las clases predefinidas y los helpers de CodeIgniter para tareas comunes.

# Contenido

1. El Modelo Vista Controlador.
2. ¿Qué es un *framework*?
3. *Frameworks* de PHP.
4. CodeIgniter.
5. Características de CodeIgniter.
6. Instalación de CodeIgniter.
7. Configuración de CodeIgniter.
8. Creación de un controlador.
9. Eliminar el index.php de las URLs.
10. Creación de métodos para el controlador.
11. Creación de vistas.

# El Modelo Vista Controlador

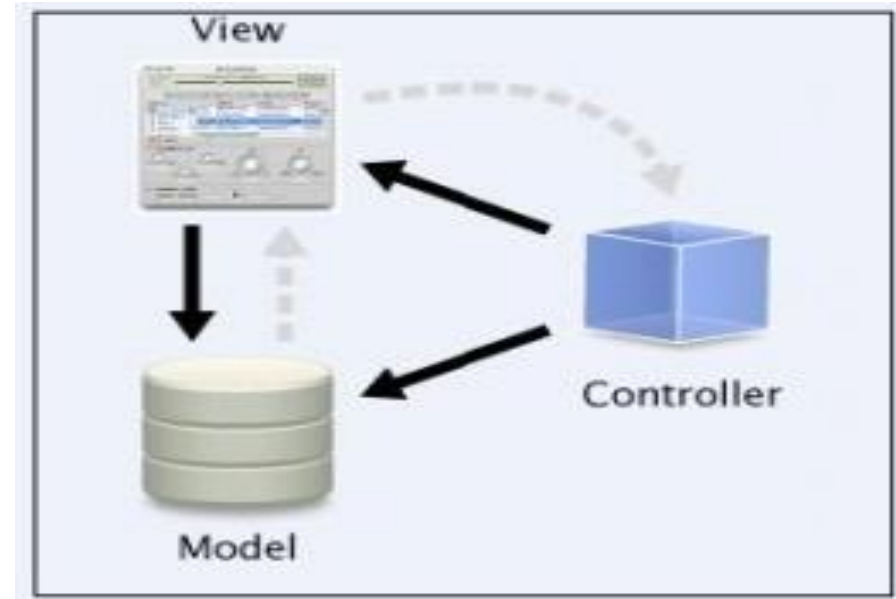
- \* Los *frameworks* basados en el Modelo Vista Controlador se han utilizado desde hace algunos años para el desarrollo de aplicaciones web del lado del servidor, CodeIgniter es uno de ellos.
- \* Es por esto que es indispensable conocer la lógica de desarrollo que hay detrás de este modelo para entenderlo bien y luego comenzar a utilizar *frameworks* basados en este modelo de desarrollo.

# El Modelo Vista Controlador

- \* El Modelo Vista Controlador es uno de los modelos de desarrollo de software más utilizado de los últimos tiempos, sobre todo en el mundo de las aplicaciones web.
- \* Este modelo fue presentado por primera vez en 1979 por Trygve Reenskaug con el lenguaje Smalltalk-76, luego fue desarrollada una biblioteca de clases con MVC para la versión de Smalltalk-80, desarrollada por Jim Althoff y otros programadores.

# El Modelo Vista Controlador

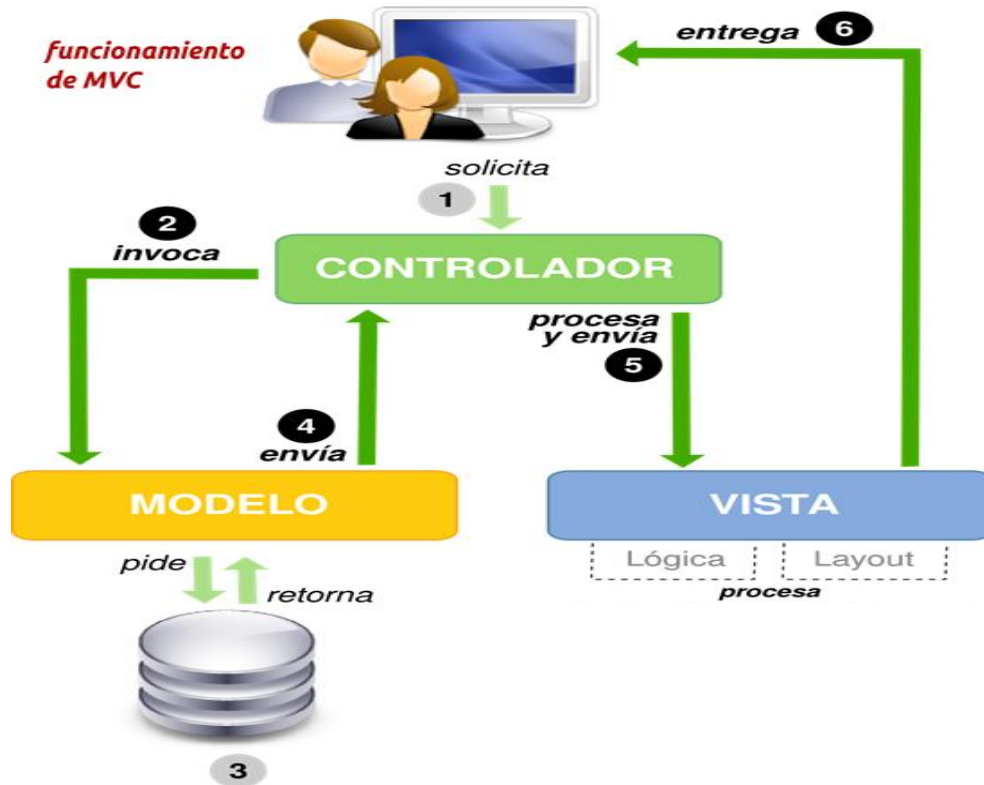
- \* Hay que indicar que MVC no es un patrón de diseño, sino un modelo que permite definir una estructura en las aplicaciones y establecer las responsabilidades y las interacciones de cada parte en dicha estructura.



# El Modelo Vista Controlador

- \* La idea que hay detrás del Modelo Vista Controlador es simple de comprender; se busca separar claramente tres aspectos que involucran el desarrollo de una aplicación web, que son:
  1. Los datos de la aplicación (el modelo),
  2. La interfaz de usuario (la vista) y
  3. La lógica de control de la aplicación (el controlador).
- \* Como se puede apreciar en la siguiente figura.

# El Modelo Vista Controlador





# El Modelo Vista Controlador

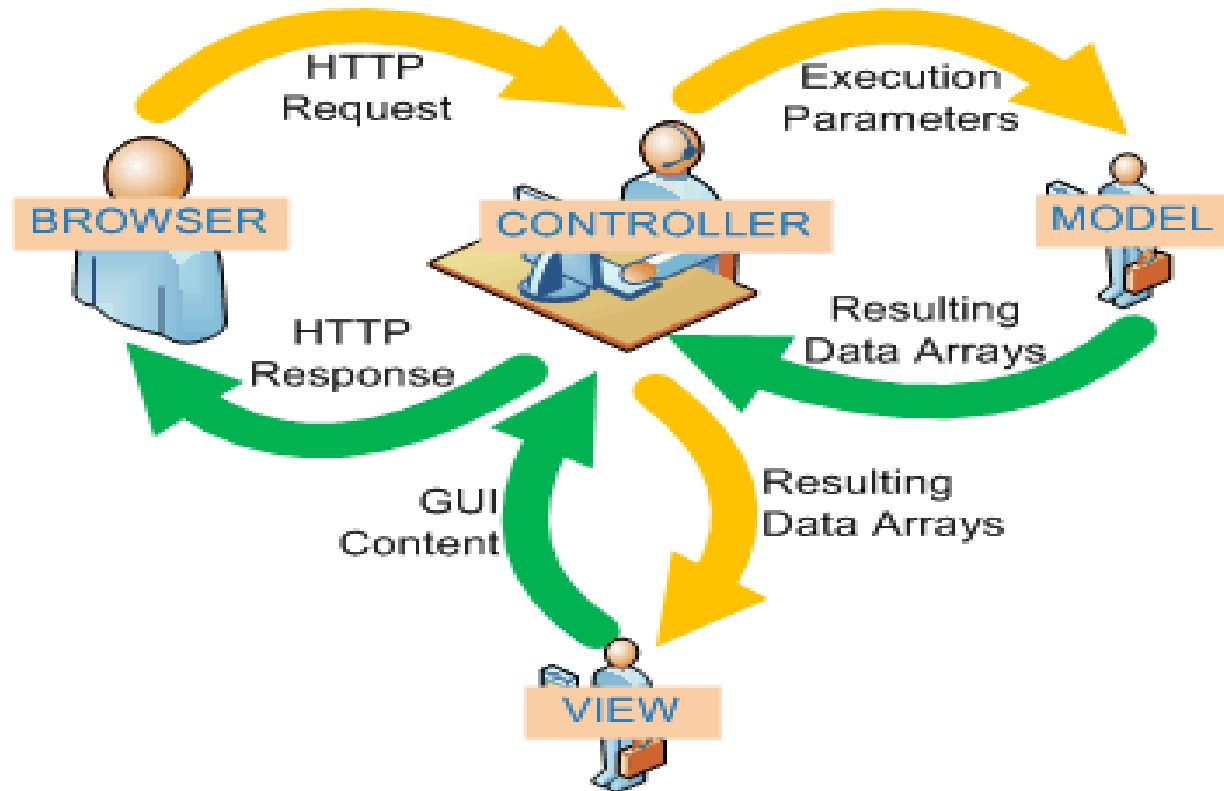
- \* En la práctica, se dice que lo que el modelo MVC busca es separar la lógica del negocio del diseño de la interfaz de usuario, facilitando la evolución por separado de ambos aspectos de desarrollo propiciando el aumento de la productividad y la reutilización.



# El Modelo Vista Controlador

- \* En el desarrollo de una aplicación web haciendo uso de este esquema de desarrollo se puede crear un modelo con todas las consultas a la base de datos que se puedan requerir de acuerdo a la lógica del negocio.
- \* Por otro lado, se pueden crear varias vistas que se mostrarán a los usuarios del sitio cada vez que soliciten una página web a través de una URL.
- \* Además, se pueden desarrollar varios controladores que gestionarán los eventos que se produzcan en la interfaz gráfica producto de la interacción de los usuarios.

# El Modelo Vista Controlador



# El Modelo Vista Controlador

- \* El flujo de control en una aplicación realizada bajo el modelo MVC es:
  1. El usuario realiza una petición en la interfaz mediante una acción que por lo regular es la realización de un clic sobre un enlace o un botón.
  2. El controlador recibe la petición y gestiona el evento de entrada.
  3. Este controlador solicita al modelo los datos que han sido solicitados mediante la acción del usuario.
  4. Una vez obtenidos los datos desde el modelo se pasan los datos a una vista para que sean mostrados al usuario.
  5. La interfaz de usuario queda a la espera de otra interacción del usuario comenzando un nuevo ciclo.

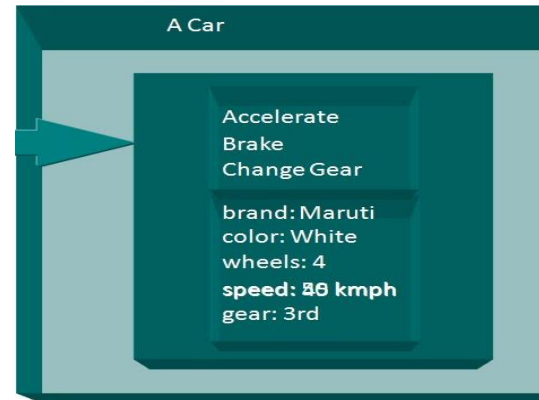
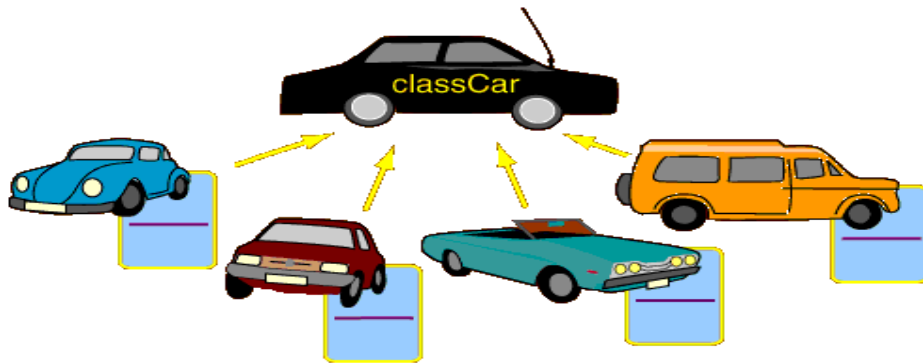
# La Vista

- \* En el mundo de las aplicaciones web, puede pensarse en **la vista** como la página **(X)HTML**, diseñada con **CSS** y controlada mediante scripts de **JavaScript** o librerías y plug-ins realizados con **jQuery**.



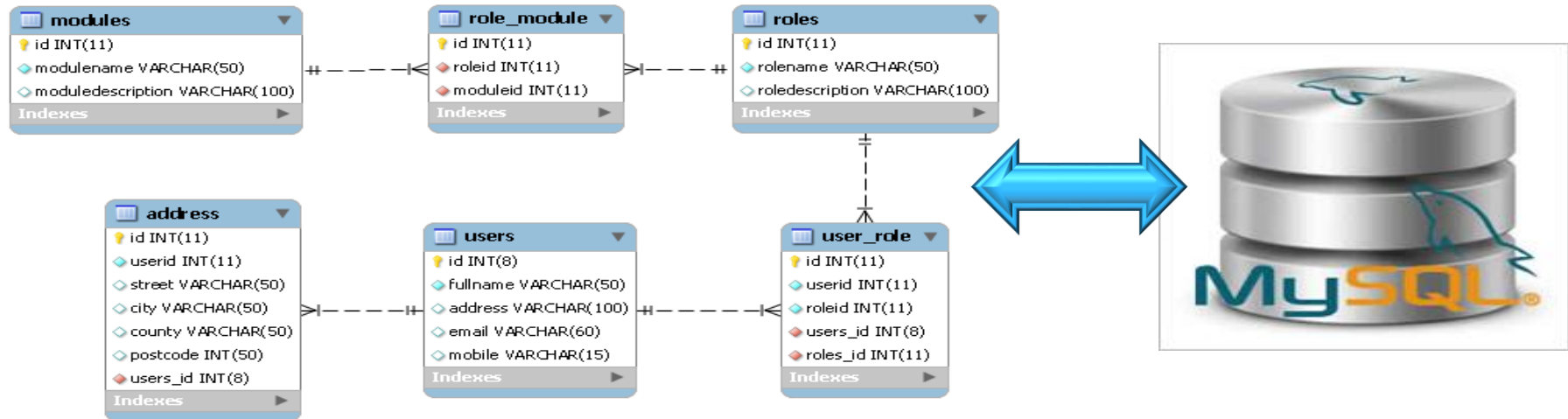
# El Controlador

- \* El **controlador** sería el código implementado mediante **clases** que se encarga de solicitar al modelo los datos que son requeridos por la acción del usuario en la interfaz gráfica, gestionando la vista que debe invocarse una vez que los datos sean obtenidos.



# El Modelo

- \* El **modelo** representaría la información almacenada en la base de datos, junto con las reglas del negocio que pueden transformar esa información con sentencias INSERT, DELETE o UPDATE.



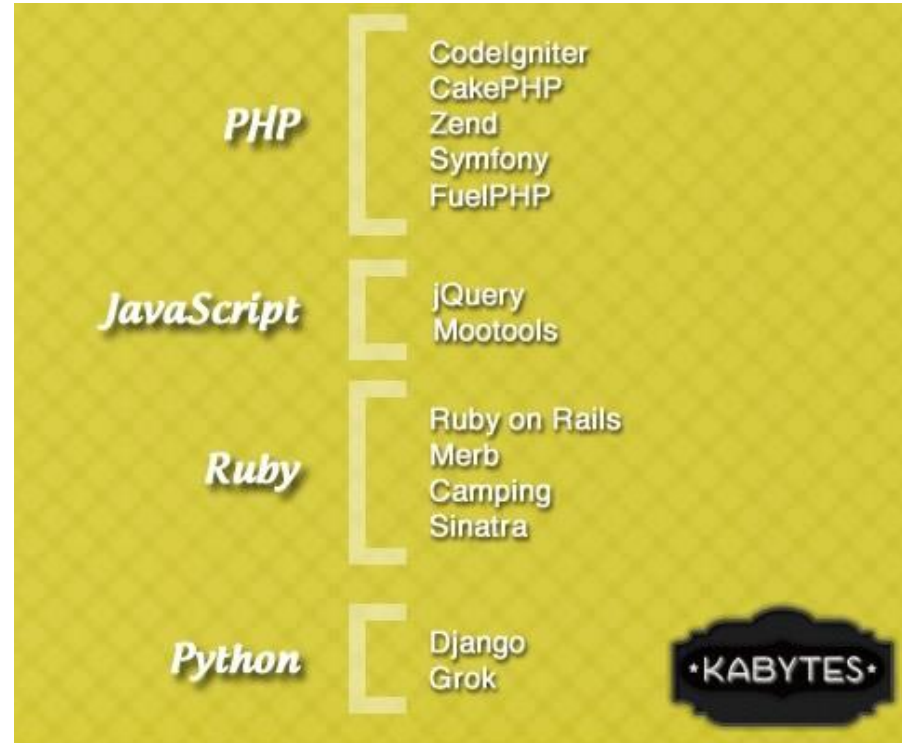
# ¿Qué es un *framework*?

- \* Conceptualmente, *framework* es un conjunto estandarizado de conceptos, prácticas y criterios utilizados para normalizar un tipo de problemática particular que sirve para enfrentar y resolver nuevos problemas de índole similar.
- \* En el desarrollo de software un *framework*, se trata de un esquema, patrón o modelo conceptual que proporciona módulos de software concretos (clases, funciones, librerías) a partir de los cuales puede desarrollarse un proyecto de software más complejo.



# ¿Qué es un *framework*?

- \* En otras palabras, un *framework* puede ser considerado como una aplicación genérica incompleta, pero que es perfectamente configurable y a la que pueden añadirse nuevos componentes para la realización de una aplicación completa.



# ¿Qué es un *framework*?

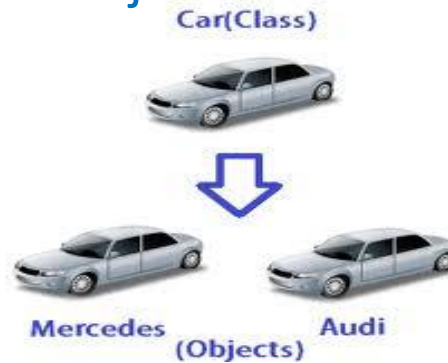
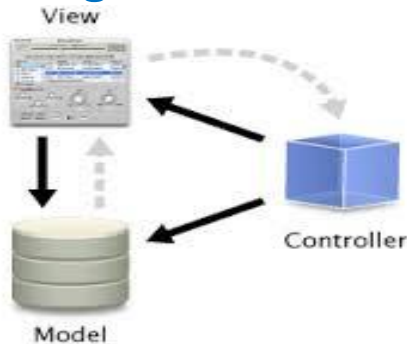
- \* Los *frameworks* son desarrollados con el objetivo de brindar a los programadores una mejor organización y estructura a los proyectos de software que realizan.
- \* Contribuyen a desarrollar aplicaciones informáticas con mayor rapidez, facilitando el mantenimiento con base en la organización de la aplicación.



# ¿Qué es un *framework*?

\* Por lo general, los *frameworks* utilizan modelos y paradigmas de programación que tienen amplio recorrido y que han sido probados suficientemente:

1. Modelo Vista Controlador.
2. Programación Orientada a Objetos.





# Frameworks de PHP

\* Existen numerosos frameworks PHP desarrollados bajo la lógica del Modelo Vista Controlador. Entre los más utilizados están:



1. Zend Framework.
2. CodeIgniter.
3. Symfony.
4. CakePHP.
5. Laravel.
6. Yii (Yes It Is!).

NOTA: Existen muchos más frameworks para PHP. Sería imposible abordarlos todos en una sola clase, así que si desea conocerlos puede visitar la dirección <http://www.phpframeworks.com>.



# Frameworks de PHP

Nombre	Descripción	Creadores
	<p>Creado para ayudar a que la producción de sitios web con PHP sea más fácil de mantener a largo plazo poniendo a disposición un conjunto estandarizado de componentes que facilitan el desarrollo.</p>	<p>El principal patrocinador de Zend Framework es Zend Technologies, no obstante muchas empresas contribuyen con componentes y características.</p>
	<p>Es un entorno de trabajo para el desarrollo de aplicaciones web en PHP que facilita y reduce el tiempo de desarrollo. Está basado en el modelo vista controlador. Se precia de ser el más rápido entre casi todos los <i>Frameworks</i> PHP.</p>	<p>Fue desarrollado por Rick Ellis, quien fundó EllisLab Inc. La empresa que le da soporte y mantenimiento a este framework.</p>

# Frameworks de PHP

Nombre (logo)	Descripción	Creadores
 The logo for Symfony, featuring the word "symfony" in a white, lowercase, sans-serif font on a dark brown rectangular background.	Completo <i>framework</i> PHP diseñado para optimizar el desarrollo de aplicaciones web basado en el modelo Vista Controlador separando lógica del negocio, de la lógica del servidor y la presentación de la aplicación.	Lanzado en el 2005 por Fabien Potencier creador de Sensio Labs, empresa francesa que provee consultoría, servicios y formación sobre tecnologías Open Source. Inicialmente se le llamó Sensio Framework.
 The logo for CakePHP, featuring a stylized 3D pie chart with a slice removed, set against a red starburst background. Below the pie chart, the text "CakePHP" is written in a bold, sans-serif font, with "Cake" in white and "PHP" in blue.	<i>Framework</i> para el desarrollo de aplicaciones web escrito en PHP, creado sobre las ideas y conceptos de Ruby on Rails. Comenzó en el 2005, luego de la popularidad alcanzada por Ruby on Rails.	Fue lanzado en abril 2005 por el programador polaco Michal Tatarynowicz. Liberando la licencia para la contribución de la comunidad de programadores. Hoy es mantenido por Cake Software Foundation Inc.

# Frameworks de PHP

Nombre (logo)	Descripción	Creadores
	Framework de código abierto creado para desarrollar aplicaciones y servicios web con PHP 5. La filosofía de este <i>framework</i> es desarrollar aplicaciones de forma elegante y simple. Laravel intenta aprovechar lo mejor de otros <i>frameworks</i> y aprovechar las características de las últimas versiones de PHP.	Fue creado en el 2011, con una gran influencia de otros <i>Frameworks</i> como Ruby On Rails, Sinatra e incluso Symfony y ASP.NET. Fue desarrollado por Taylor Otwell.
	Framework de código abierto, orientado a objetos, cuyas siglas Yii (Yes It Is!) significan ¡Sí lo es!. Surge como alternativa a PRADO, superando muchos de los problemas presentados por este Framework. Yii es mucho más fácil y eficiente que PRADO.	Fue lanzado oficialmente el 3 de diciembre del 2008. Actualmente, se dispone de la versión 2.0.11. La última versión de la rama 1, está en la versión 1.1.18, lanzada el 31 de diciembre del 2016. Este Framework es desarrollado por la compañía Yii Software LLC.

# CodeIgniter

- \* CodeIgniter es un *framework* diseñado bajo el modelo MVC (Modelo Vista Controlador) que facilita el desarrollo de aplicaciones web con lenguaje PHP.
- \* A pesar de que existen muchos *frameworks* PHP, CodeIgniter destaca del resto por la rapidez, su extensibilidad y lo sencillo de aprender que resulta. A pesar de esto es posible desarrollar poderosas aplicaciones si se sabe utilizar bien.



# CodeIgniter

- \* CodeIgniter provee de un numeroso conjunto de bibliotecas de clases para tareas usuales en una aplicación web como: manejo de URLs, creación y validación de formularios, uso de sesiones, paginación de resultados, acceso a bases de datos, etc.



# Características de CodeIgniter

- \* Es **libre**: esto significa que el Framework está liberado bajo licencia tipo Apache/BSD de software Open Source.
- \* Es **liviano**: el núcleo del sistema únicamente requiere unas pocas bibliotecas muy pequeñas. Esto lo distingue de otros frameworks que vienen cargados de muchos más recursos.
- \* **Carga dinámica** de bibliotecas: si se requieren bibliotecas adicionales, estas se pueden cargar en tiempo de ejecución, lo que contribuye a que el sistema sea ligero y bastante rápido.

# Características de CodeIgniter

- \* Es **rápido**: realmente es muy difícil encontrar un framework PHP que sea más rápido en respuesta que CodeIgniter, esto se debe a lo liviano de las bibliotecas bases.
- \* Usa **MVC**: utiliza el enfoque del Modelo Vista Controlador que permite la separación entre la lógica del negocio y la presentación.
- \* Posee **URLs claras**: las URLs generadas por CodeIgniter son claras y amigables con los motores de búsqueda, utilizando el enfoque basado en segmentos:

`http://dominio.com/controlador/metodo/parametro.`

# Características de CodeIgniter

- \* Es **extensible**: el sistema se puede extender fácilmente a través de sus propias bibliotecas, asistentes (*helpers*), extensiones de clase o sistema de *hooks* .
- \* No requiere de un **motor de plantillas**: aunque provee de un sencillo motor de plantillas, que puede utilizar de forma opcional, CodeIgniter no obliga a usarlo. Esto se debe a que los motores de plantillas no pueden igualar el desempeño de PHP nativo. Además, aprender a utilizar un motor de plantillas suele ser un tanto más complicado que aprender la sintaxis de PHP.

# Instalación de CodeIgniter

- \* Lo primero que debe hacer para comenzar a utilizar CodeIgniter es descargarlo del sitio oficial de EllisLab Inc. Y luego instalarlo en su servidor web.
- \* La descarga se realiza desde la dirección:  
`https://codeigniter.com/download`
- \* En la actualidad ya se encuentra libre para descarga la versión 3.0 de CodeIgniter, así como la versión 2 cuyo desarrollo ha llegado hasta la versión 2.2.

# Instalación de CodeIgniter



CodeIgniter comes in two main flavors: CodeIgniter 3.x and CodeIgniter 2.x

## CodeIgniter 3.0

CodeIgniter 3.0.0 is the current version of the framework.

There have been a number of refinements since version 2.x, notably with the database, session handling and encryption. Development of this version is ongoing.

[View Code On Github](#)

[Download CodeIgniter 3.0](#)

[Download the User Guide to read offline](#)

[Download the User Guide in epub format](#)

[Download the system message translations](#)

## CodeIgniter 2.x

CodeIgniter 2.2.2 is the legacy version of the framework.

The 2.x branch was originally released January 2011, the next major update (2.1.0) came in the fall of 2011, version (2.2.0) came out in July, 2014, and the current version (2.2.2) came out in April, 2015.

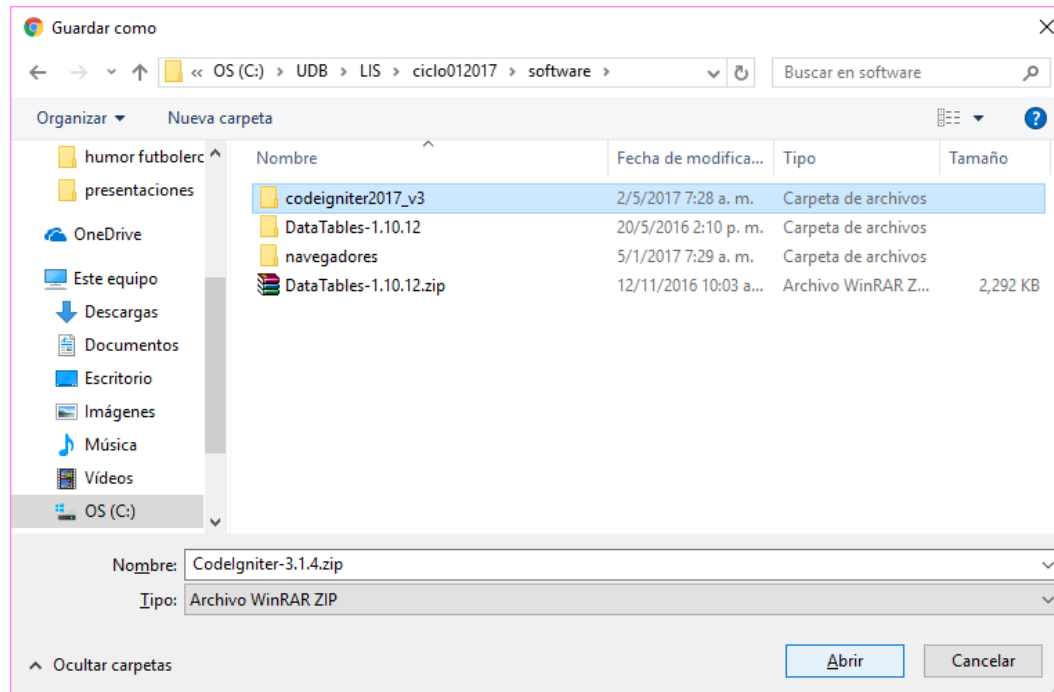
[View Code On Github](#)

[Download CodeIgniter 2.2 \(stable\)](#)

Clic en  
Descargar  
(Download)

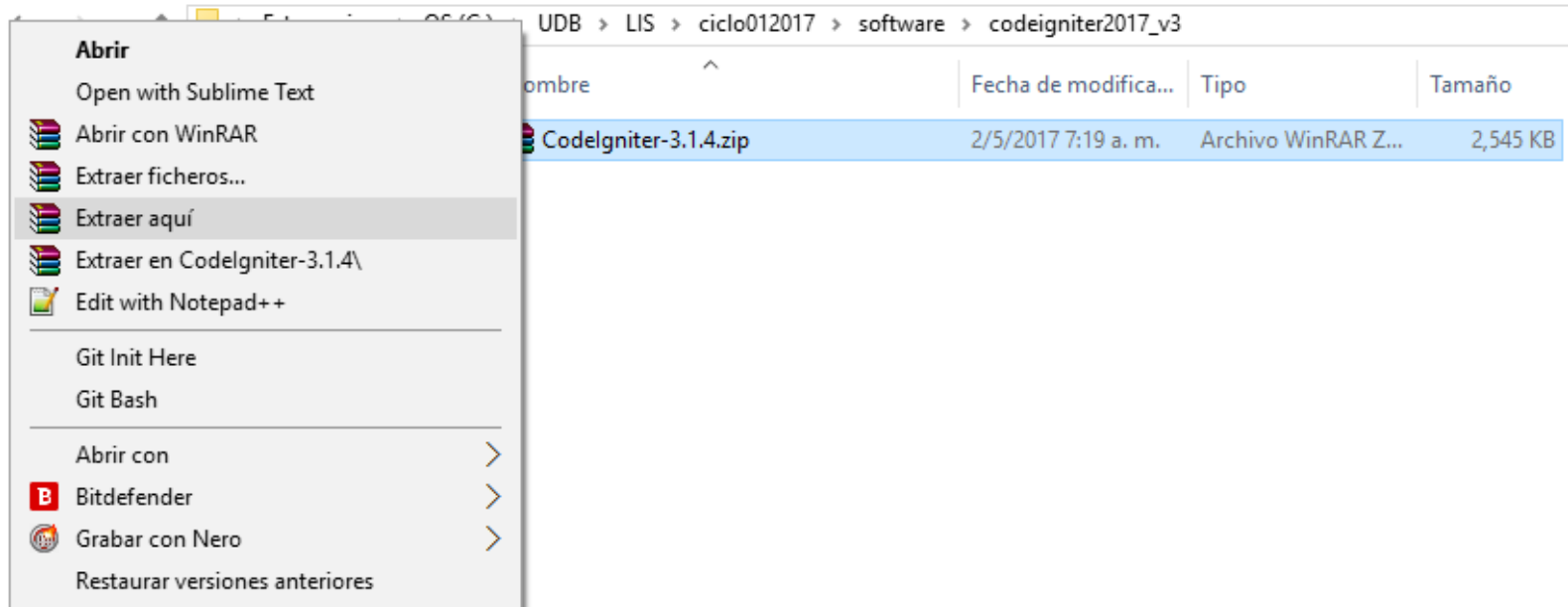
# Instalación de CodeIgniter

- \* Guarde el archivo en alguna carpeta de su equipo local:



# Instalación de CodeIgniter

- \* Descomprima el archivo de CodeIgniter con el programa de su elección.



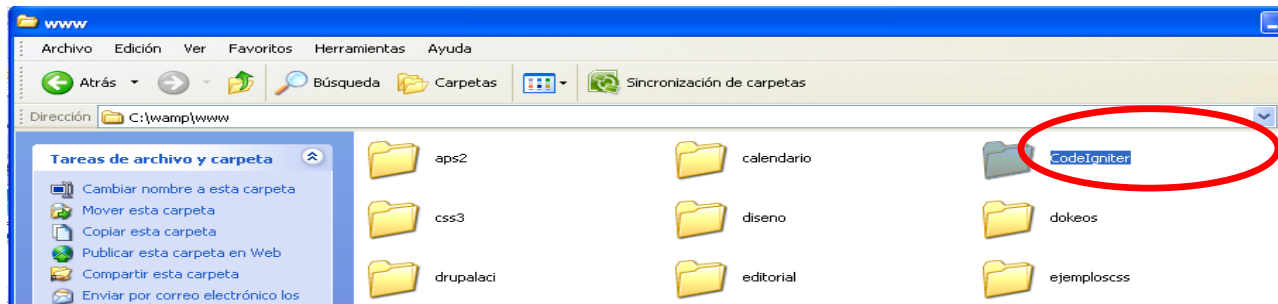


# Instalación de CodeIgniter

- \* Renombre la carpeta con los archivos extraídos y nómbrala CodeIgniter.

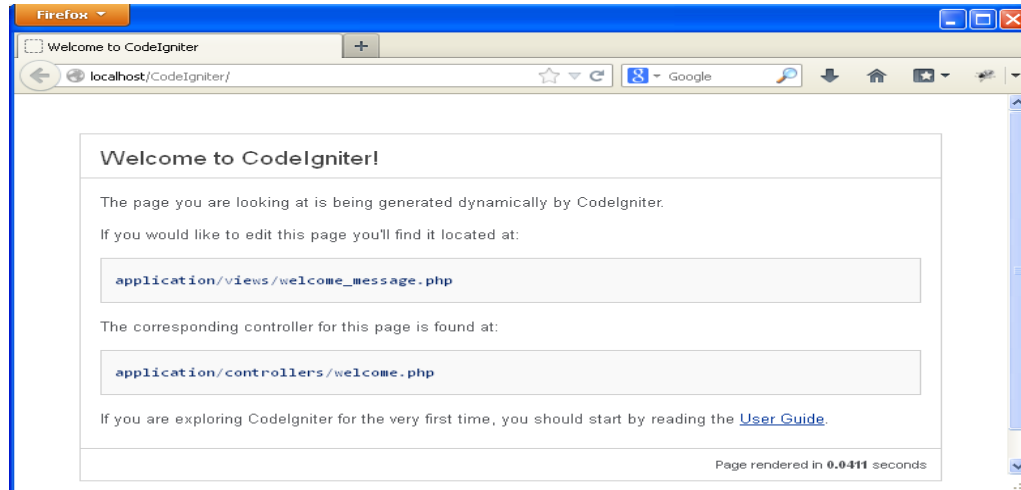


- \* Copie la carpeta completa al directorio (carpeta) web de su servidor. En el caso de Wamp esta carpeta es www.



# Instalación de CodeIgniter

- \* Para probar si CodeIgniter está correctamente instalado en su servidor, debe abrir un navegador e ingresar la URL siguiente:
- \* `http://localhost/CodeIgniter/`

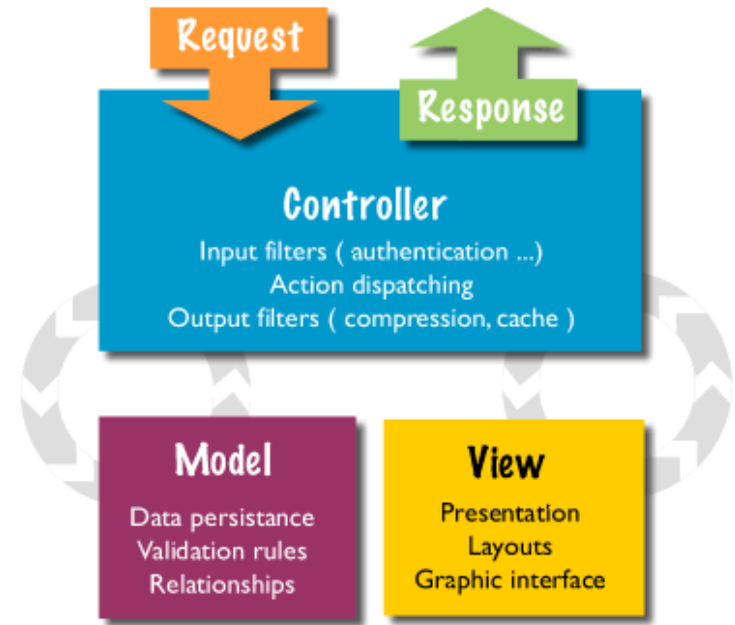


# Instalación de CodeIgniter

- \* Dentro de la carpeta de instalación de CodeIgniter encontrará dos carpetas importantes, una llamada **application** y la otra llamada **system**.
- \* La carpeta **system** es la carpeta que podríamos considerar el núcleo de CodeIgniter.
- \* La carpeta **application** es donde crearemos nuestras aplicaciones, principalmente trabajaremos dentro de las carpetas **controllers**, **models** y **views**.
- \* En la carpeta **controllers** crearemos los controladores, en **models**, los modelos y en **views** nuestras vistas.

# Instalación de CodeIgniter

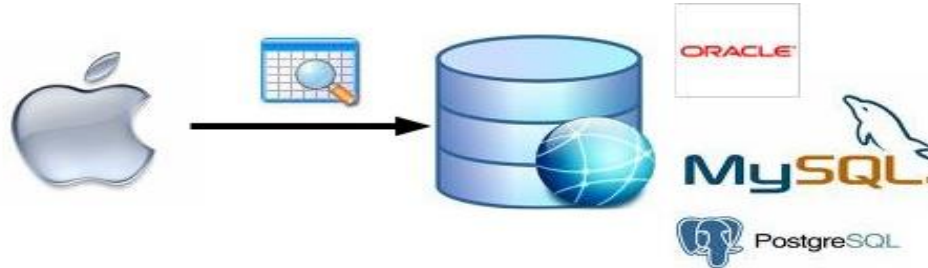
- \* En los **controladores** se crearán clases que manejarán la lógica de la aplicación, auxiliándose de los modelos para obtener información de una base de datos, si esta es solicitada o cargando las vistas con dichos datos para mostrarlos al usuario en el navegador.



Akelos Framework MVC Diagram

# Instalación de CodeIgniter

- \* En los **modelos** se definen todas las operaciones que tienen que ver con el acceso a la base de datos, manteniendo encapsulada toda la complejidad de la lógica del negocio.
- \* En esta capa se crean las clases o funciones que permiten obtener, agregar, actualizar o eliminar datos de las tablas de la base de datos.



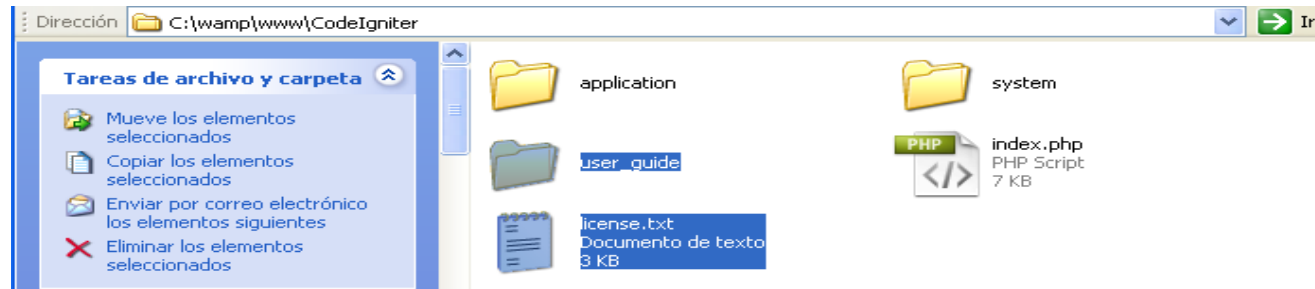
# Instalación de CodeIgniter

- \* Las **vistas** contienen los guiones (scripts) que constituyen las páginas o fragmentos de páginas web con las que interactuará el usuario. En otras palabras, las vistas contendrán principalmente código (X)HTML, CSS y JavaScript que será el encargado de generar la página web que se mostrará a los usuarios.



# Configuración de CodeIgniter

- \* Si bien es cierto, CodeIgniter ya está instalado, es preciso realizar algunas configuraciones iniciales para poder trabajar con aplicaciones que accedan a bases de datos y que manejen sus propios controladores.
- \* Primero que nada debe eliminar de los archivos extraídos del paquete CodeIgniter todo lo que no es necesario, por ejemplo la guía de usuario y el contrato de licencia.



# Configuración de CodeIgniter

- \* El siguiente paso debe ser ingresar a los archivos de la carpeta config en su instalación de CodeIgniter.
- \* Básicamente, la configuración inicial implica editar algunas líneas de los siguientes archivos:
  - \* `autoload.php`
  - \* `config.php`
  - \* `database.php`
  - \* `routes.php`



# Creación de un controlador

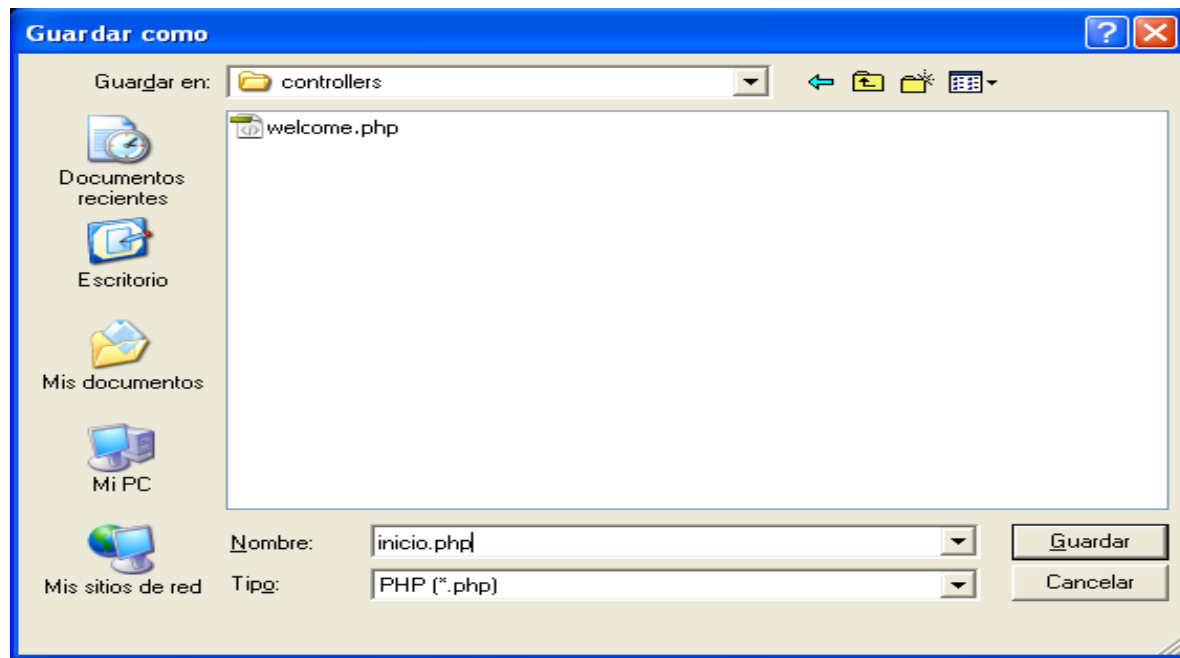
- \* Por ser un *framework* basado en el Modelo Vista Controlador (MVC), lo primero que se hace es crear un controlador.
- \* Los controladores de CodeIgniter se guardan en la carpeta *controllers* que puede encontrar dentro de la carpeta *application*.
- \* Para crear un pequeño controlador de ejemplo, puede comenzar por digitar el siguiente código en su editor PHP preferido.

# Creación de un controlador

```
<?php
class Inicio extends CI_Controller {
    function __construct(){
        parent::__construct();
    }
    function index(){
        echo "<h2>Bienvenidos a CodeIgniter framework de
PHP.</h2>";
    }
}
?>
```

# Creación de un controlador

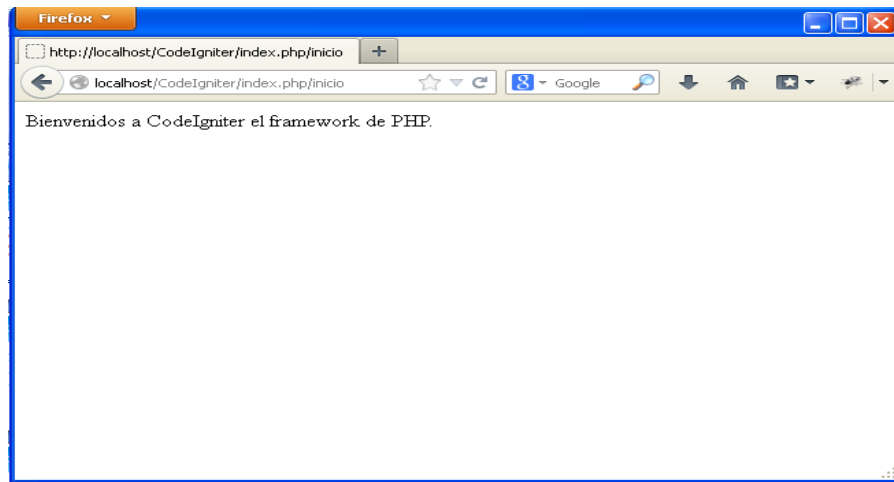
- \* Guarde el código anterior con el nombre inicio.php dentro de la carpeta controllers.



# Creación de un controlador

- \* Para visualizar el controlador que se acaba de crear, tendrá que abrir un navegador y acceder a la siguiente URL:

`http://localhost/CodeIgniter/index.php/inicio.`



# Eliminar el index.php

- \* Si bien es cierto, las URLs de CodeIgniter son muy claras y amigables, resulta inconveniente para los visitantes ingresar una URL tan larga.
- \* Un proceso que se puede seguir de forma opcional es eliminar la necesidad de utilizar el index.php como parte de las URLs.
- \* Para esto se requiere crear un archivo .htaccess que cambiará la configuración sobre el uso del index.php en las URLs de CodeIgniter.

# Eliminar el index.php

- \* Ingrese el siguiente código en su editor de texto preferido:

```
<IfModule mod_rewrite.c>  
    RewriteEngine on  
    RewriteCond $1 !^(index.php|css|js|images|img|robots.txt)  
    RewriteRule ^(.*)$  
/lis/ciclo012016/guias/guia12/ejemplo2/index.php/$1 [L]  
</IfModule>
```

- \* El archivo anterior debe ser guardado en la carpeta o directorio raíz de la instalación del CodeIgniter con el nombre .htaccess.

# Eliminar el index.php

- \* Como siguiente paso cambiaremos una opción de configuración dentro del archivo config.php de CodeIgniter, que se localiza dentro de la carpeta config dentro de **application**.
- \* Hay que editar el archivo para que la opción de configuración `$config['index_page']` quede establecida como cadena vacía y no como index.php.

```
$config['index_page'] = '';
```

# Eliminar el index.php

- \* Adicionalmente, debe modificar la directiva de configuración `$config['base_url']` siempre dentro del archivo de configuración `config.php`.

```
$config['base_url'] = 'http://localhost/CodeIgniter/';
```

OBSERVACIÓN: La ruta real es la ruta de instalación de CodeIgniter en su servidor.



# Establecer el controlador por defecto

- \* Para cambiar el controlador por defecto que viene pre-configurado con CodeIgniter debe abrir el archivo routes.php que encontrará dentro de la carpeta **config** en **application**.
- \* Ubíquese en las líneas siguientes dentro del código:

```
$route['default_controller'] = "welcome";  
$route['404_override'] = '';
```

- \* Realice la siguiente modificación:

```
$route['default_controller'] = "inicio";  
$route['404_override'] = '';
```

# Establecer el controlador por defecto

- \* El efecto que tendrá la modificación anterior es que ahora ya no necesitará ingresar en la barra de direcciones del navegador lo siguiente:

`http://localhost/CodeIgniter/inicio`

- \* Únicamente, bastará con digitar la siguiente URL:

`http://localhost/CodeIgniter/`

# Creación de métodos para el controlador

- \* Dentro del controlador, que es una clase, se pueden incorporar métodos para funciones específicas.
- \* Para esto sólo es necesario abrir el archivo de nuestro controlador e incorporar métodos, así como se hace en cualquier clase estándar de PHP.

# Creación de métodos para el controlador

```
class Inicio extends CI_Controller {
    function __construct() { parent::__construct(); }
    function index(){}
    //Métodos del controlador
    function materia(){
        $data['titulo'] = "Vista de CodeIgniter";
        $data['subtitulo'] = "Lenguajes Interpretados en el Servidor";
        $data['contenidos'] = array("Sintaxis básica", "Tipos de datos",
"Estructuras de Control", "Funciones", "Objetos");
        $this->load->view("inicio_view", $data);
    }
    function mensaje($dato){
        echo $dato;
    }
}
```

# Creación de métodos para el controlador

- \* Para visualizar los mensajes, debe abrir el navegador de su preferencia e ingresar las siguientes URLs:

`http://localhost/CodeIgniter/inicio/materia`

`http://localhost/CodeIgniter/inicio/mensaje/Saludos`

# Creación de vistas

- \* Las vistas son las páginas que los usuarios van a cargar en el navegador. Son la interfaz de usuario final creada con la estructura que todo documento HTML bien construido debe poseer.
- \* Como todo documento HTML, hoy en día, además de código HTML, puede incluir hojas de estilo (CSS), guiones o librerías (scripts) de JavaScript, jQuery, bootstrap y cualquier otro elemento válido para un documento web.

# Creación de vistas

- \* Para hacer un uso estricto del Modelo Vista Controlador, el contenido a mostrar en las páginas debe ser generado en las vistas y no en el controlador como lo hemos hecho en el ejemplo anterior.
- \* Por lo tanto, cualquier sentencia echo o printf debe ser gestionada en la vista y no en el controlador.
- \* Las vistas se crearán en la carpeta views dentro de application donde se instaló el CodeIgniter.

# Creación de vistas

- \* El aspecto de una vista es justo como el de una página web, donde además de HTML, al tratarse de una página dinámica del lado del servidor, requerirá muy probablemente, código PHP.
- \* Observe por ejemplo, el siguiente código de una vista.



# Creación de vistas

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Uso de una vista</title>
  <link rel="stylesheet" href="<?php echo BASE ?>css/estilos.css">
  <script src="<?php echo BASE ?>js/modernizr.custom.js"></script>
</head>
<body>
<header>
  <h1>Vista de CodeIgniter</h1>
</header>
<section>
  <article>
    <h2>Vistas</h2>
    <p>
      La vista es la que se muestra al usuario y es cargada a través del controlador.
      Esto significa que cuando el usuario ingresa una URL es el controlador quien la
      toma, no se carga la vista de forma automática. Una vez el controlador analiza
      la URL, se encarga de buscar la vista asociada al controlador.
    </p>
```

# Creación de vistas

- \* Para poder cargar nuestra vista, tendremos que modificar el controlador anterior, quitando la sentencia echo que colocamos cuando mostramos el ejemplo.
- \* La modificación se muestra a continuación:

# Creación de vistas

```
class Inicio extends CI_Controller {
    function __construct()
    {
        parent::__construct();
    }
    function index()
    {
        //echo "Bienvenidos a CodeIgniter el framework de PHP.";
    }

    //Métodos del controlador
    function materia() {
        $this->load->view("inicio_view");
    }

    function mensaje($dato) {
        echo $dato;
    }
}
```

# Creación de vistas

- \* Puede notar que en el método materia de nuestro controlador Inicio, se agregó una sentencia:  

```
$this->load->view("nombre_vista");
```
- \* Es así como se invocan las vistas desde el controlador. Cada método creado en el controlador podría invocar una vista diferente en una aplicación más compleja.
- \* Para llamar a la vista desde el navegador debería ingresar la siguiente dirección:  
<http://localhost/CodeIgniter/inicio/materia>

# Creación de vistas

- \* En la práctica, casi siempre necesitará mandar datos desde el controlador hacia la vista para que sean mostrados.
- \* Es decir, el contenido de una vista pueden ser datos que gestione el controlador, que mediante un mecanismo definido por CodeIgniter serán enviados para que sean mostrados en la página de nuestra vista.
- \* Observe el código del siguiente controlador:

# Creación de vistas

```
class Inicio extends CI_Controller {
    function __construct()
    {
        parent::__construct();
    }
    function index(){}

    //Métodos del controlador
    function materia(){
        $data['titulo'] = "Vista de CodeIgniter";
        $data['subtitulo'] = "Lenguajes Interpretados en el Servidor";
        $data['contenidos'] = array("Sintaxis básica", "Tipos de datos",
        "Estructuras de Control", "Funciones", "Objetos");
        $this->load->view("inicio_view", $data);
    }
}
```

# Creación de vistas

- \* Como puede observar, se crea un array asociativo llamado \$data en el que se van definiendo como índices las variables que utilizará la vista para mostrar los datos que le son asignados a este array.
- \* La particularidad es que a este array se le define un índice que también es otro array, con los elementos definidos como cadenas de texto. También podrían ser otros tipos de datos en lugar de cadenas.

# Creación de vistas

- \* Lo verdaderamente importante es que la vista puede utilizar los índices del array \$data como variables, para simplificar el manejo.
- \* Ahora bien, en la vista habría que hacer cambios para mostrar la información que es enviada desde el controlador.
- \* El código modificado de la vista se muestra a continuación:



# Creación de vistas

```
<header>
    <h1><?php echo $titulo ?></h1>
</header>
<section>
    <article>
        <h2><?php echo $subtitulo ?></h2>
        <ul>
            <?php
                foreach($contenidos as $itemdata):
                    echo "\t<li>" . $itemdata . "</li>\n";
                endforeach;
            ?>
        </ul>
    </article>
</section>
```

# Creación de modelos

- \* En el ejemplo anterior, los datos pasados a la vista se crearon directamente dentro del método del controlador.
- \* En la mayor parte de los casos se trabajará con datos procedentes de una base de datos que para aplicaciones con PHP, con frecuencia será una base de datos MySQL, aunque podría perfectamente ser otro tipo de base de datos también.

# Creación de modelos

- \* Ahora mostraremos cómo crear un modelo que obtenga los datos de la base de datos y cómo son gestionados estos datos desde el controlador para que sean pasados a la vista para su visualización en la página web creada.
- \* Lo primero que hay que hacer cuando se va a crear un modelo utilizando CodeIgniter es configurar el archivo database.php que se encuentra en la carpeta config.

# Creación de modelos

- \* Lo que se hará en esta configuración es básicamente proporcionar los datos de conexión con el servidor MySQL (host, usuario, contraseña, driver de conexión y base de datos con la que se trabajará).
- \* Esto se puede visualizar en las siguientes líneas del archivo de configuración database.php.

# Creación de modelos

```
$db['default']['hostname'] = 'localhost';  
$db['default']['username'] = 'root';  
$db['default']['password'] = '';  
$db['default']['database'] = 'libros';  
$db['default']['dbdriver'] = 'mysqli';  
$db['default']['dbprefix'] = '';  
$db['default']['pconnect'] = TRUE;  
$db['default']['db_debug'] = TRUE;  
$db['default']['cache_on'] = FALSE;  
$db['default']['cachedir'] = '';  
$db['default']['char_set'] = 'utf8';  
$db['default']['dbcollat'] = 'utf8_general_ci';  
$db['default']['swap_pre'] = '';  
$db['default']['autoinit'] = TRUE;  
$db['default']['stricton'] = FALSE;
```

# Creación de modelos

- \* El siguiente paso es indicar que queremos cargar automáticamente la librería database en el archivo de configuración autoload.php que también está almacenado en la carpeta config de su instalación de CodeIgniter.
- \* Realmente, este paso no es necesario hacerlo acá. Lo que sucede es que casi siempre desarrollará aplicaciones que guardan o accedan a información que procede de bases de datos. En otros casos, puede cargar la librería en el controlador donde se requiera una determinada librería.

# Creación de modelos

- \* En el índice asociativo 'libraries' de la matriz \$autoload se indica mediante asignación de un array con una entrada que para el caso de esta librería será 'database'. Tal como se indica en el comentario, se pueden agregar más librerías, separándolas por comas.

```
/*  
| Prototype:  
|$autoload['libraries'] = array('database','session','xmlrpc');  
*/  
$autoload['libraries'] = array('database');
```

# Creación de modelos

- \* En el siguiente ejemplo se crea una sentencia SQL dentro de un modelo para recuperar los datos del ISBN, autor, título y precio de la tabla libros.
- \* La consulta creada es pasada como argumento del método query de la librería db que fue cargada automáticamente en el archivo autoload.php.
- \* Luego, utilizando el método result() se puede recorrer por cada registros de resultados utilizando una sentencia repetitiva.



# Creación de modelos

```
<?php
class libros_model extends CI_Model{
    function mostrarLibros(){
        //Construir la consulta a ejecutar en el servidor MySQL
        $sql = "SELECT isbn, autor, titulo, precio ";
        $sql .= "FROM libros";
        //Ejecutar la consulta llamando al método query del objeto db
        $qr = $this->db->query($sql);
        if($qr->num_rows() > 0){
            foreach($qr->result() as $row){
                $data[] = $row;
            }
            return $data;
        }
    }
}
?>
```

# Creación de modelos

- \* El modelo debe almacenarse en la carpeta models de su instalación de CodeIgniter, que está ubicada también dentro de application.
- \* A continuación, se debe crear un controlador que cargue el modelo y obtenga los datos que son requeridos. Luego deberá invocarse a la vista para que muestre los datos obtenidos desde el modelo.

# Creación de modelos

```
<?php
class libros extends CI_Controller{
    function index(){
        //Cargar el modelo
        $this->load->model('libros_model');
        //Invocar al método del modelo que obtendrá los datos
        $data['rows'] = $this->libros_model->mostrarLibros();
        //Cargar la vista y pasarle los datos del modelo
        $this->load->view('libros_view', $data);
    }
}
?>
```

# Creación de modelos

- \* Por último, la vista `libros_view` que mostraría los datos que le ha pasado el controlador `libros` a través de la matriz `$data["rows"]`.
- \* Note que la vista tomará los datos directamente del índice `"rows"` que es una matriz en sí misma.
- \* La vista tendrá que extraer los datos de la matriz refiriéndola directamente como `$rows` y recorriendo sus elementos con alguna estructura repetitiva conveniente.

# Creación de modelos

```
<?php
    foreach($rows as $book):
        echo "<tr>";
        echo "<td>";
        echo $book->isbn;
        echo "</td>";
        echo "<td>";
        echo $book->autor;
        echo "</td>";
        echo "<td>";
        echo $book->titulo;
        echo "</td>";
        echo "<td>";
        echo $book->precio;
        echo "</td>";
        echo "</tr>";
    endforeach;
?>
```

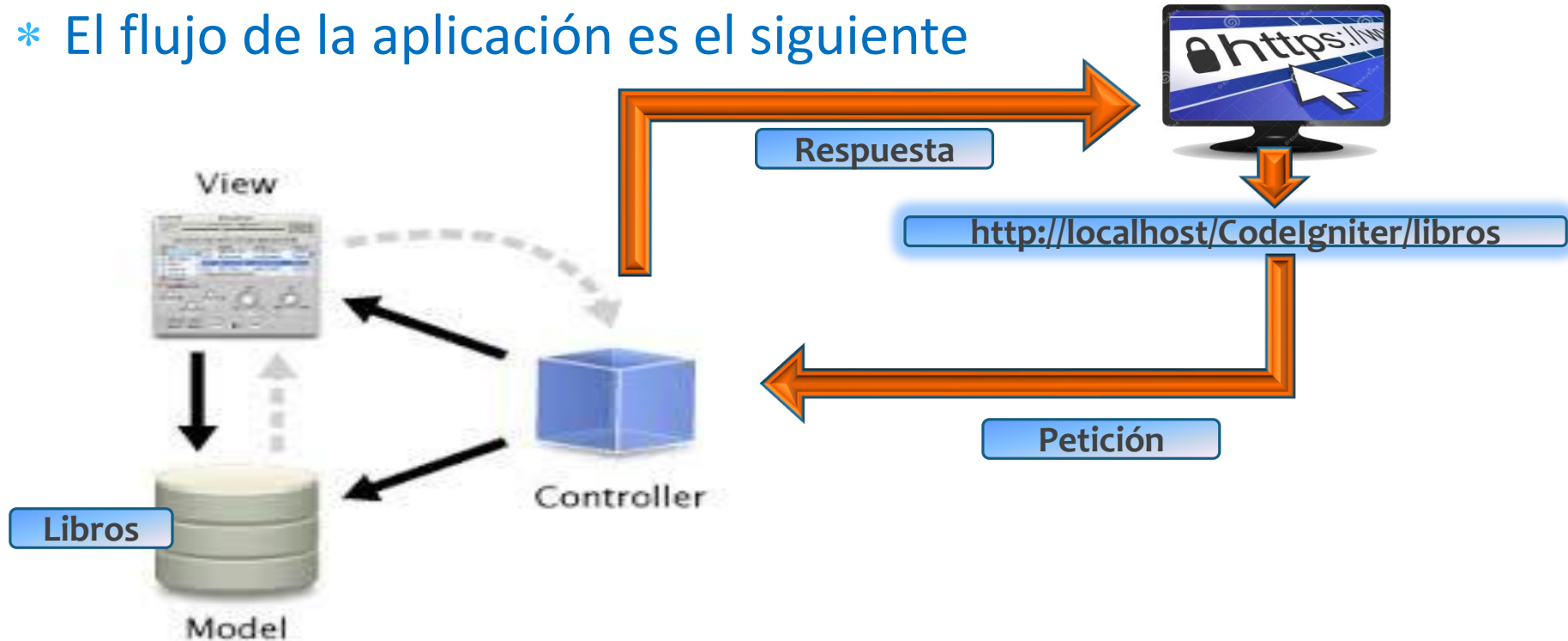
# Creación de modelos

- \* Para poner en marcha el controlador que se auxiliará del modelo `libros_model` para obtener los datos de la base de datos libros y luego pasárselos a la vista debe ingresar la siguiente URL:

`http://localhost/CodeIgniter/libros`

# Creación de modelos

- \* El flujo de la aplicación es el siguiente



# Utilizar Active Record

- \* Con CodeIgniter es posible usar una versión modificada del patrón de base de datos conocido como Active Record.
- \* Este patrón permite obtener, insertar y actualizar información en una base de datos con mínima codificación. En algunas ocasiones, sólo una o dos líneas de código serán necesarias para realizar una acción en la base de datos.



# Utilizar Active Record

- \* Entre los beneficios de utilizar Active Record destaca el permitir que la aplicación sea independiente de la base de datos que utiliza, ya que la sintaxis de consulta es generada por cada adaptador de base de datos.
- \* Además, proporciona consultas más seguras, ya que los valores que se pasan a las consultas son filtrados y escapados automáticamente por el sistema.

# Seleccionar datos con Active Record

- \* Los siguientes métodos permiten construir una sentencia SELECT de SQL.
- \* `$this->db->get()`. Ejecuta la consulta de selección y devuelve el resultado. Sólo debe ser utilizado cuando se desean obtener todos los registros de una tabla con todos sus campos.

\* Ejemplo:

```
$qr = $this->db->get('libros');  
//Equivale a SELECT * FROM libros;
```

# Seleccionar datos con Active Record

- \* Además del nombre de la tabla, se pueden pasar hasta 3 argumentos al método `get()`. Estos argumentos deben ser datos numéricos enteros que representan cláusulas de límite y principio en una consulta.

- \* Por ejemplo:

```
$qr = $this->db->get('libros',10,25);  
//Equivale a SELECT * FROM libros LIMIT 25,10  
//En MySQL. En otras bases de datos, tiene  
//alguna diferencia.
```

# Seleccionar datos con Active Record

- \* Notará que en ambos casos, los resultados son devueltos a la variable `$qr`. Esta variable se convierte en un objeto que mediante el método `result()` nos permitirá mostrar los resultados usando una sentencia repetitiva.

- \* Por ejemplo:

```
$qr = $this->db->get('libros') ;  
foreach($qr->result() as $row) {  
    echo $row->isbn . "|" . $row->titulo . "|" . $row->  
    >autor;  
}
```

# Seleccionar datos con Active Record

- \* Al modificar el modelo del ejemplo libros\_model para utilizar Active Record tendríamos el siguiente código:

```
<?php
class libros_model extends CI_Model{
    function mostrarLibros(){
        //Consulta usando Active Record
        $qr = $this->db->get('libros');
        if($qr->num_rows() > 0){
            foreach($qr->result() as $row){
                $data[] = $row;
            }
            return $data;
        }
    }
}
?>
```

# Seleccionar datos con Active Record

- \* No sería necesario modificar ni el controlador ni la vista.
- \* Como puede notar la reducción de líneas de código es considerable, teniendo en cuenta que no necesita construir la consulta, ya que el método solo requiere el nombre de la tabla que se quiere consultar, es el gestor de base de datos el que construye la consulta, por eso la independencia de éste al utilizar Active Record.

# Otras opciones para seleccionar datos

- \* `$this->db->get_where()`. Es idéntica a `$this->db->get()`, con la única diferencia que permite agregar una cláusula `WHERE` en el segundo argumento del método.

- \* Ejemplo:

```
$qr = $this->db->get_where('libros',array('titulo LIKE'=>
    '%JavaScript%')) ;
//Equivale a SELECT * FROM libros WHERE titulo LIKE
//'%JavaScript%'
```

# Otras opciones para seleccionar datos

- \* `$this->db->where()`. También es posible ejecutar separadamente los métodos `get()` y `where()` si eso le resulta más claro y menos confuso.

- \* Ejemplo:

```
$qr = $this->db->where(array('titulo LIKE' => '%PHP%'));  
$qr = $this->db->get('libros');  
//Note que primero se coloca el where() y después el método get()  
//Equivale a SELECT * FROM libros WHERE titulo LIKE  
// '%PHP%'
```

- \* De forma alternativa puede utilizar:

```
$qr = $this->db->where(array('titulo LIKE' => '%PHP%'))  
->get('libros');
```



# Especificar la lista de campos a consultar

- \* `$this->db->select()`. Permite indicar en la consulta la lista de campos que se desean recuperar. Esta función no debería ser utilizada si desea consultar todos los campos (`SELECT *`), en ese caso es mejor utilizar `$this->db->get()`.

## \* Ejemplo:

```
$qr = $this->db->select('titulo, autor')  
      ->get('libros');  
//Equivale a SELECT titulo, autor FROM libros
```

# Consultar datos de varias tablas

- \* Es posible realizar consulta de datos que provengan de distintas tablas utilizando la cláusula JOIN de SQL, siempre desde el enfoque Active Record.

- \* Ejemplo:

```
$qr = $this->db->select('titulopelicula, generopelicula, nombre,
    duracion')
    ->from('peliculas')
    ->join('genero', 'peliculas.idgenero =
genero.idgenero')
    ->join('director', 'peliculas.iddirector =
director.iddirector')
    ->get();
```

# Consultar datos de varias tablas

\* La consulta anterior equivale a:

```
SELECT peliculas.titulopelicula, genero.generopelicula,  
       director.nombre, peliculas.duracion  
FROM peliculas  
JOIN genero  
     ON peliculas.idgenero = genero.idgenero  
JOIN director  
     ON peliculas.iddirector = director.iddirector;
```

# Insertar datos en tablas

- \* Active Record de CodeIgniter también proporciona métodos para insertar registros en cualquier tabla de la base de datos.
- \* Se pueden utilizar dos métodos para insertar registros en tablas. Estos métodos son `$this->db->insert()` y `$this->db->insert_batch()`.
- \* En ambos métodos los valores que son pasados son escapados de forma automática para producir consultas más seguras.

# Insertar datos en tablas

- \* `$this->db->insert()`. Genera una consulta SQL de inserción basado en los datos proporcionados al método, ejecutando a la vez la consulta.
- \* Se pueden pasar un arreglo o un objeto a esta función.
- \* El único cambio está en la construcción del array o la del objeto con los nombres de los campos y los valores que les serán asignados.

# Insertar datos en tablas

\* Ejemplo con array.

```
$databook = array(  
    "isbn" => "978-97-010-7275-2",  
    "autor" => "Catherine M. Ricardo",  
    "titulo" => "Bases de Datos",  
    "precio" => 42.75  
);  
$this->db->insert('libros', $databook);  
//Equivale a INSERT INTO libros(isbn,autor,titulo,precio)  
//VALUES('978-97-010-7275-2','Catherine M. Ricardo',  
//'Bases de Datos',42.75)
```

# Insertar datos en tablas

\* Ejemplo con objeto.

```
class book{  
    private $isbn = "978-97-010-7275-2";  
    private $autor = "Catherine M. Ricardo";  
    private $titulo = "Bases de Datos";  
    private $precio = 42.75  
}  
$databook = new book();  
$this->db->insert('libros', $databook);  
//Equivale a INSERT INTO libros(isbn,autor,titulo,precio)  
//VALUES('978-97-010-7275-2','Catherine M. Ricardo',  
//'Bases de Datos',42.75)
```

# Insertar datos en tablas

- \* `$this->db->insert_batch()`. Genera una sentencia INSERT de SQL basada en los argumentos proporcionados y ejecuta la consulta.
- \* Esta función también acepta un array o un objeto como argumento.
- \* La diferencia con el método anterior `insert()` es que en este método se pueden pasar múltiples arrays, representando cada uno un registro diferente.



# Insertar datos en tablas

## \* Ejemplo con array.

```
$databook = array(  
    array("isbn" => "978-84-415-3151-2",  
        "autor" => "David Sawyer McFarland",  
        "titulo" => "JavaScript y jQuery",  
        "precio" => 99.75),  
    array("isbn" => "978-84-415-2711-9",  
        "autor" => "David Sawyer McFarland",  
        "titulo" => "CSS",  
        "precio" => 68.60)  
);  
$this->db->insert('libros', $databook);  
//Equivale a INSERT INTO libros(isbn,autor,titulo,precio)  
//VALUES('978-84-415-3151-2','David Sawyer McFarland',  
//'JavaScript y jQuery',42.75), ('978-84-415-2711-9',  
//'David Sawyer McFarland',68.60)
```

# Actualizar datos en tablas

- \* Para poder modificar datos de un registro que ya está insertado en la tabla se puede utilizar el método `update` del objeto `db`.
- \* `$this->db->update()`. Genera una consulta SQL de actualización basado en los datos suministrados. El método `update()` acepta un array o un objeto como argumento.

# Actualizar datos en tablas

## \* Ejemplo con array.

```
$databook = array(  
    "autor" => "Catherine Marie Ricardo",  
    "titulo" => "Sistemas de Bases de Datos",  
    "precio" => 56.20  
);  
$this->db->where('isbn', '978-97-010-7275-2');  
$this->db->update('libros', $databook);  
//Equivale a UPDATE libros  
//SET autor='Catherine Marie Ricardo',  
//titulo='Sistemas de Bases de Datos', precio=42.75  
// WHERE isbn='978-97-010-7275-2';
```

# Actualizar datos en tablas

## \* Ejemplo con objeto.

```
class book{
    private $autor = "Catherine Marie Ricardo";
    private $titulo = "Sistemas de Bases de Datos";
    private $precio = 56.20
}
$databook = new book;
$this->db->where('isbn', '978-97-010-7275-2');
$this->db->update('libros', $databook);
//Equivale a UPDATE libros
//SET autor='Catherine Marie Ricardo',
//titulo='Sistemas de Bases de Datos', precio42.75
//WHERE isbn='978-97-010-7275-2';
```

# Eliminar datos de las tablas

- \* La última de las operaciones de manipulación de datos es la eliminación. Se trata de la sentencia DELETE de SQL que permitirá borrar registros de la base de datos.
- \* `$this->db->delete()`. Genera una sentencia SQL de eliminación basado en los argumentos proporcionados ejecutando a la vez la consulta.

```
$this->db->delete('libros', array('isbn'=>'978-97-010-7275-2'))
```

```
//Equivale a DELETE FROM libros WHERE isbn='978-97-010-7275-2'
```

# Eliminar datos de las tablas

- \* Una particularidad del método delete() es que permite que sea pasado un arreglo de tablas, lo que le posibilitará eliminar registros de más de una tabla a la vez, siempre y cuando el criterio de la cláusula WHERE sea cumplido en todas las tablas.

```
$tablas = array('libros', 'revistas', 'manuales');  
$this->db->where('autor', 'Catherine Marie Ricardo');  
$this->db->delete($tablas);
```

# Eliminar datos de las tablas

- \* Para eliminar todos los registros de una tabla se pueden usar los métodos `empty_table()` y `truncate()`. El primero de los métodos es equivalente a una sentencia `DELETE FROM tabla`, en tanto que el segundo es equivalente a una sentencia `TRUNCATE tabla`.
- \* Recuerde que la diferencia entre `DELETE` y `TRUNCATE` es que la primera elimina la estructura de la tabla y los datos, en tanto que la segunda sólo elimina los datos de la tabla.

```
$this->db->empty_table('libros');  
//Equivale a DELETE FROM libros  
$this->db->truncate('libros');  
//Equivale a TRUNCATE libros
```

# FIN

Desarrollo de Aplicaciones Web con  
Software Interpretado en el Servidor