UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

# Proffessional backend developer

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
**www.sceu.frba.utn.edu.ar/e-learning**

# Modulo 1: Introducción y nivelación

# Unidad 3: Codeigniter 3 (Parte 1)

# Presentación:

En la presente unidad veremos las ventajas que nos presenta el uso de frameworks bajo el patrón MVC como por ejemplo Codeigniter.

El patrón MVC es utilizado en la mayoría de los frameworks de PHP de hoy en día.

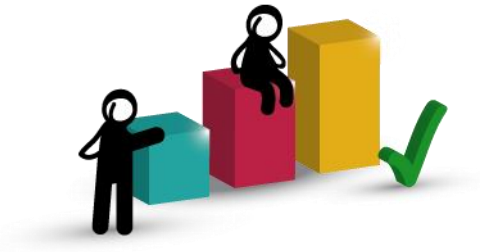
# Objetivos:

## Que los participantes*:

- Aprendan el uso del patrón MVC en PHP
- Configuren codeigniter para uso
- Entiendan apliquen las ventajas propuestas por Codeigniter.

## Bloques temáticos*:

- Patrón MVC
- Codeigniter – Instalación y configuración
- Codeigniter – Modelo, Vista y Controlador
- Codeigniter – Base de datos
- Codeigniter – URL



## Consignas para el aprendizaje colaborativo

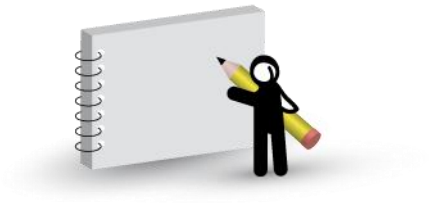
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



# Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

# Patrón MVC

## Definición

Es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

## Historia

- Fue introducido por Trygve Reenskaug (web personal) en Smalltalk-76 durante su visita a Xerox Parc6 en los años 70 y, seguidamente, en los años 80, Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80.8
- En 1988, MVC se expresó como un concepto general en un artículo sobre Smalltalk-80.Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos

## Definición actual

El ‘controlador’ es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el ‘modelo’ y la ‘vista’, y unifica la validación (utilizando llamadas directas o el “observer” para desacoplar el ‘modelo’ de la ‘vista’ en el ‘modelo’ activo)

Derivados del MVC:

- HMVC (MVC Jerárquico)
- MVA (Modelo-Vista-Adaptador)
- MVP (Modelo-Vista-Presentador)
- MVVM (Modelo-Vista Vista-Modelo)

## Modelo

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

Envía a la ‘vista’ aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al ‘modelo’ a través del ‘controlador’.

## Controlador

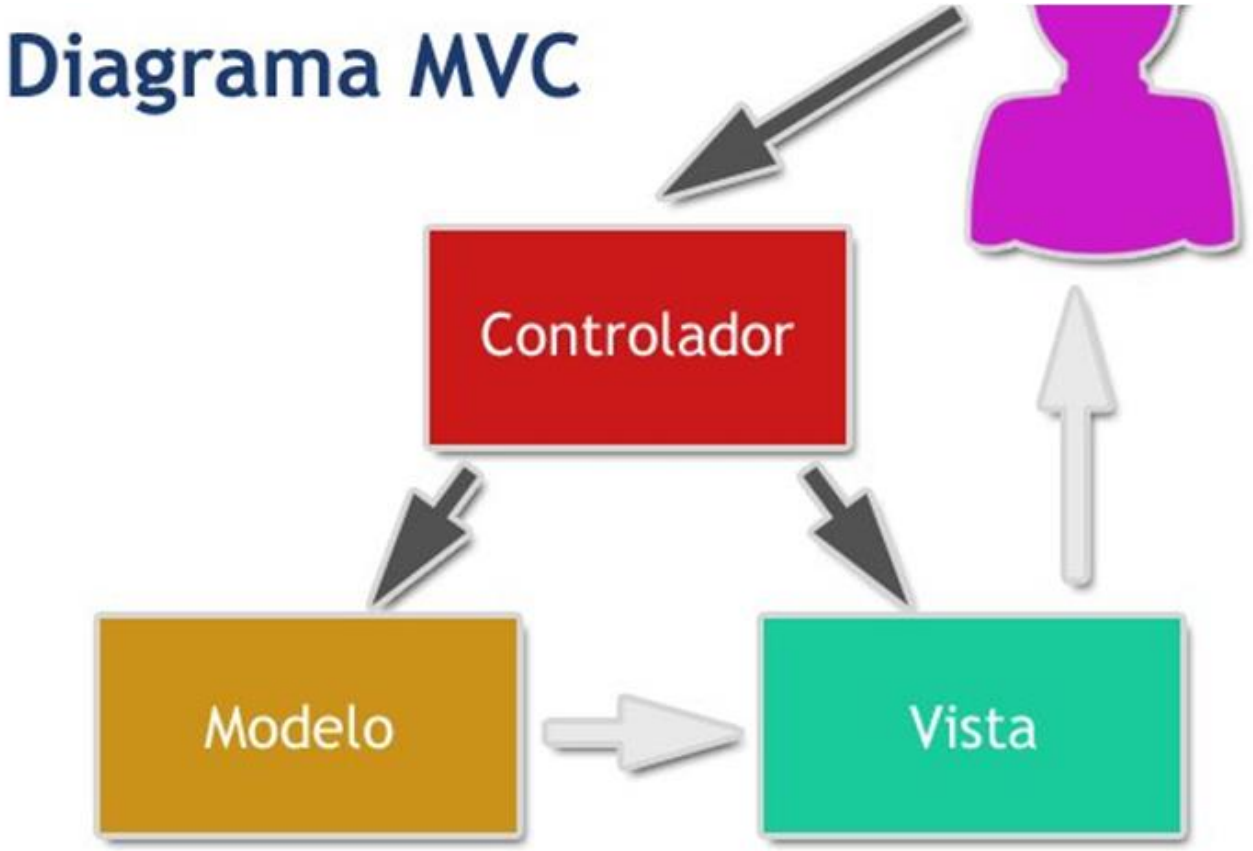
Responde a eventos (usualmente acciones del usuario) e invoca peticiones al ‘modelo’ cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos).

También puede enviar comandos a su ‘vista’ asociada si se solicita un cambio en la forma en que se presenta el ‘modelo’ (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el ‘controlador’ hace de intermediario entre la ‘vista’ y el ‘modelo‘.

## Vista

Presenta el ‘modelo’ (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho ‘modelo’ la información que debe representar como salida.

## Esquema



## Interacción de ejemplo

- El usuario realiza una solicitud a nuestro sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.
- El controlador comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
- Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación; por eso esa flecha la hemos coloreado de otro color.
- Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.

## Lógica de ejemplo

Es un conjunto de reglas que se siguen en el software para reaccionar ante distintas situaciones.

En una aplicación el usuario se comunica con el sistema por medio de una interfaz, pero cuando acciona esa interfaz para realizar acciones con el programa, se ejecutan una serie de procesos que se conocen como la lógica del negocio.

También tiene normas sobre lo que se puede hacer y lo que no se puede hacer (reglas de negocio)

**La lógica del negocio queda del lado de los modelos.**

Por ejemplo, pensemos en un sistema que implementa usuarios.

Los usuarios pueden realizar comentarios.

Pues si en un modelo nos piden eliminar un usuario nosotros debemos borrar todos los comentarios que ha realizado ese usuario también.

Eso es una responsabilidad del modelo y forma parte de lo que se llama la lógica del negocio.

## Lógica de aplicación

Es algo que pertenece a los controladores.

Por ejemplo, cuando me piden ver el resumen de datos de un usuario.

Esa acción le llega al controlador, que tendrá que acceder al modelo del usuario para pedir sus datos.

Luego llamará a la vista apropiada para poder mostrar esos datos del usuario.

**Todo ese conjunto de acciones que se realizan invocando métodos de los modelos y mandando datos a las vistas forman parte de la lógica de la aplicación.**

**Ejercicio propuesto**

Tenemos que desarrollar una aplicación para una distribuidora de bebidas.

El objetivo es tener un sistema de control de stock, contamos con las siguientes tablas en la base de datos:

- Usuarios
- Productos
    - Nombre
    - Descripción
    - Precio
    - Stock
    - Categoría

Categorías Debemos tener en cuenta las siguientes reglas de negocio:

- El nombre del producto debe ser obligatorio
- La descripción debe ser menor a 255 caracteres
- El precio debe ser mayor a 0 (cero)
- El producto debe tener una categoría.

**¿Cada una de ellas se debe introducir en el modelo, controlador o vista?**

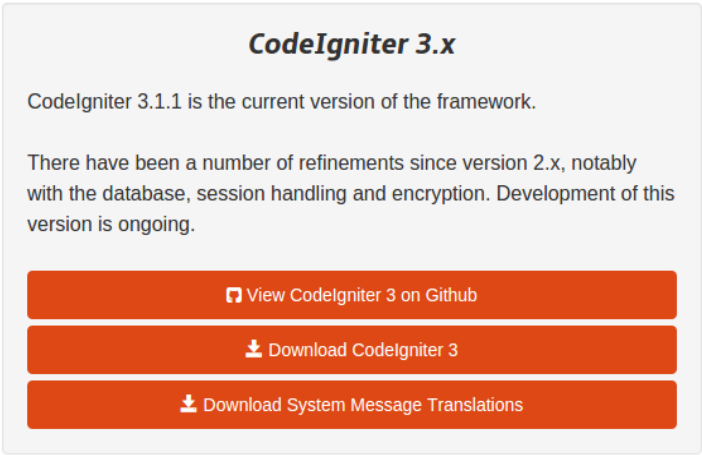
# Codeigniter – Instalación y configuración

## Instalación

Codeigniter es un framework en PHP que utiliza el patrón MVC.

**Lo podes descargar desde: https://www.codeigniter.com/download**

**En este ejemplo trabajaremos con la versión 3.x**



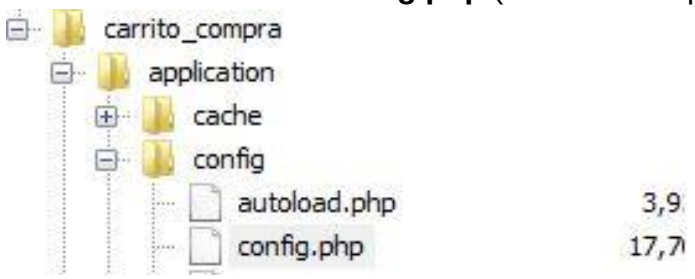

Una vez descargado:

- Descomprimirlo y pegarlo en htdocs


- Renombrar la carpeta con el nombre del proyecto

## Config.php

- Una vez descargado:
  - • Abrir el archivo **config.php** (ubicado en application->config)




  - Modificar **base_url**, debemos colocar la url que tendrá nuestro sitio (de acuerdo al nombre colocado en el paso anterior)

```
|
| If it is not set, then CodeIgniter will try guess the protocol and path
| your installation, but due to security concerns the hostname will be set
| to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.
| The auto-detection mechanism exists only for convenience during
| development and MUST NOT be used in production!
|
| If you need to allow multiple domains, remember that this file is still
| a PHP script and you can easily do that on your own.
|
*/
$config['base_url'] = 'http://localhost/carrito_compras/';
```

## database.php

- Configurar el archivo **database.php** (ubicado en application->config).
- Debemos introducir la configuración de nuestra base de datos

```
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => '',
    'password' => '',
    'database' => '',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

- Ahora accederemos a la pantalla de bienvenida de **codeigniter**


## Autoload

En el **autoload** (application->config->autoload.php) podremos indicar las bibliotecas que deseamos que se carguen “automáticamente” en nuestra aplicación.

**En este caso database y sesión**

```
/*
| -------------------------------------------------------------------
|  Auto-load Libraries
| -------------------------------------------------------------------
| These are the classes located in system/libraries/ or your
| application/libraries/ directory, with the addition of the
| 'database' library, which is somewhat of a special case.
|
| Prototype:
|
|   $autoload['libraries'] = array('database', 'email', 'session');
|
| You can also supply an alternative library name to be assigned
| in the controller:
|
|   $autoload['libraries'] = array('user_agent' => 'ua');
*/
//$autoload['libraries'] = array();
$autoload['libraries'] = array('database', 'session');

/*
```

# Route

En el archivo **application->config->route.php** podremos configurar el controlador por defecto que queremos que tenga nuestra aplicación.

```
| This is not exactly a route, but allows you to automatically route
| controller and method names that contain dashes. '-' isn't a valid
| class or method name character, so it requires translation.
| When you set this option to TRUE, it will replace ALL dashes in the
| controller and method URI segments.
|
| Examples: my-controller/index -> my_controller/index
|       my-controller/my-method -> my_controller/my_method
*/
$route['default_controller'] = 'index_productos';
```

# Codeigniter – Modelo, Vista y Controlador

## Modelo

Los modelos están definidos en **application->models**. El nombre de la clase debe ser el mismo nombre del archivo.


Ejemplo de modelo:

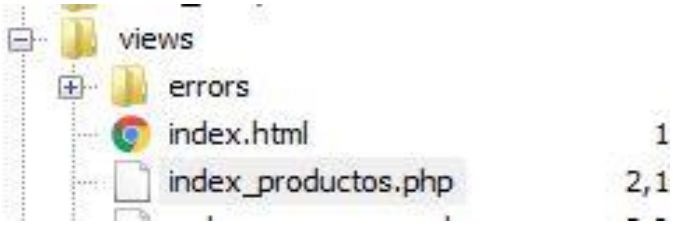
```php
<?php
class productos_model extends CI_Model  {

    function __construct(){
        parent::__construct();
    }

    function productos(){
        $this->db->select('id, precio');
        $this->db->from('compras_productos');
        $consulta = $this->db->get();
        $resultado = $consulta->result();
        return $resultado;
    }

}
?>
```

Para agregar un modelo debemos crear un archivo en el directorio mencionado. Se debe respetar tener el mismo nombre en el archivo que en la clase (no poder el _model en el nombre del archivo).

La clase debe heredar de la clase **CI_Model**

# Vista

Los archivos están ubicados en **Application->views**. Podrá tener cualquier nombre, **se debe indicar todo en minúscula**.




Ejemplo:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
?><!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Welcome to CodeIgniter</title>

    <style type="text/css">

    ::selection { background-color: #E13300; color: white; }
    ::-moz-selection { background-color: #E13300; color: white; }

    body {
        background-color: #fff;
        margin: 40px;
        font: 13px/20px normal Helvetica, Arial, sans-serif;
        color: #4F5155;
    }

    a {
        color: #003399;
        background-color: transparent;
        font-weight: normal;
    }
```

Para agregar una vista debemos crear la misma en el directorio mencionado. La extensión de la vista debe ser .php

# Controlador

Los controladores estarán ubicados en:

**Application->controllers.**

El nombre de la clase del controlador debe ser igual al nombre del archivo (Con la primera letra mayúscula)


Ejemplo:

```php
class Index_productos extends CI_Controller {

    /**
     * Index Page for this controller.
     *
     * Maps to the following URL
     *      http://example.com/index.php/welcome
     *  - or -
     *      http://example.com/index.php/welcome/index
     *  - or -
     * Since this controller is set as the default controller in
     * config/routes.php, it's displayed at http://example.com/
     *
     * So any other public methods not prefixed with an underscore will
     * map to /index.php/welcome/<method_name>
     * @see https://codeigniter.com/user_guide/general/urls.html
     */
    public function index()
    {
        $this->load->view('index_productos');
    }
}
```

Para agregar un controlador debemos crear el archivo con el mismo nombre que la clase (primera letra en mayúscula).

El controlador debe heredar de **CI_Controller.**

# Cargar una vista desde el controlador

Para llamar al contenido de una vista desde el controlador, debemos hacerlo de la siguiente manera:

**$this->load->view(nombre_vista, parametros)**

Nombre_vista: es el nombre del archivo creado en la carpeta views, sin el .php

Parametros: array de parámetros enviado a la vista.

Por ejemplo:

```php
//Llama a la vista que genera el listado de productos
$parametros = array();
$parametros["productos_destacados"] = $productos;
$this->load->view('productos_destacados',$parametros);
```

En el ejemplo anterior vemos como se carga la vista desde el controlador, por parámetro se envía un array con los productos consultados al modelo.

El array que se pasa por parámetro ($parametros) tiene una clave “productos_destacados”, la vista recibirá esa clave como una variable:

```php
<div class="row xsResponse categoryProduct">
    <?php
        //Iteramos los productos destacados para mostrarlos en la home del sitio
        foreach($productos_destacados as $k => $v){
    ?>
    <!-- LISTADO DE PRODUCTOS DESTACADOS -->
    <div class="item itemauto col-lg-3 col-md-3 col-sm-6 col-xs-6">
        <div class="product">
            <a class="add-fav tooltipHere" data-toggle="tooltip" data-original-title="Add to
            Wishlist"
               data-placement="left">
                <i class="glyphicon glyphicon-heart"></i>
            </a>
```

# Cargar un modelo desde el controlador

Desde el controlador debemos llamar al siguiente método:

**$this->load->model(nombre_modelo);**

Luego para llamar a un método del modelo debemos hacerlo de la siguiente manera:

**$this->nombre_modelo->metodo();**

Por ejemplo:

```
public function index()
{
    $this->load->model('productos_model');
    $productos = $this->productos_model->productos();
    var_dump($productos); exit;
    $this->load->view('index_productos');
}
```

En el ejemplo vemos como se carga el modelo “productos_model” y se consulta al método “productos” de dicho modelo.

# Codeigniter – Base de datos

## Insertar

Para insertar un registro en la base de datos desde nuestro modelo, debemos llamar al objeto db y al método **insert().** Este método recibirá dos parámetros, el nombre de la tabla y un array asociativo en donde los índices serán el nombre de las tablas y los valores son los datos a guardar:

```
$data = array(
    'titulo' => 'Título del informe',
    'descripcion' => 'Descripción del informe',
    'prioridad' => 3
);
$this->db->insert('informes', $data);
```

## Actualizar

El método update sirve para modificar un registro existente.

Con el método **where** establecemos la condición para la modificación.

```
$data = array(
    'titulo' => 'Título del informe',
    'descripcion' => 'Descripción del informe',
    'prioridad' => 3
);
$this->db->where('id', 1);
$this->db->update('informes', $data);
```

El primer parámetro del método update es la tabla que queremos modificar, el segundo parámetro es el array de datos a modificar.

## Ultimo id insertado

Cuando insertamos un registro nuevo, mediante este método podemos obtener el id en el que se inserto.

```
$id = $this->db->insert_id();
```

## Eliminar

Llamamos el método **delete** en el cual pasamos como parámetro la tabla en donde queremos eliminar el registro.

Con el **where** establecemos la condición para realizar la eliminación.

```
$this->db->where('id', 1);
$this->db->delete('informes');
```

## Select

```
$this->db->select('id, titulo, descripcion, prioridad');
$this->db->from('informes');
$consulta = $this->db->get();
$resultado = $consulta->result();
```

En este caso vemos lo siguiente:

- Select: recibe las columnas que queremos obtener en la consulta.
- From: indica la tabla que vamos a consultar
- Get: Realiza la consulta a la base de datos
- Result: Obtiene los resultados consultados. **Si utilizamos result_array nos devolverá los resultados en un array en lugar de una colección de objetos.**

En caso de querer retornar una sola fila (un único resultado) debemos realizar lo siguiente.

```
$this->db->select('id, titulo, descripcion, prioridad');
$this->db->from('informes');
$this->db->where('id', $id);
$consulta = $this->db->get();
$resultado = $consulta->row();
return $resultado;
```

## Join

```
$this->db->select('*');
$this->db->from('publicaciones');
$this->db->join('comentarios', 'comentarios.publicacion_id = publicaciones.id');
$consulta = $this->db->get();
$resultado = $consulta->result();
```

A la sentencia vista anteriormente le agregamos la sentencia **join**.

El primer parámetro indica la tabla con la que queremos realizar el join, el segundo parámetro indica las condiciones del **ON** del join. En caso de querer realizar un left join debemos pasar un tercer parámetro que indique 'left'.

## Consulta simple

```
$consulta = $this->db->query("SELECT * FROM informes");
$resultado = $consulta->result();
```

Mediante el método query podemos realizar cualquier tipo de consulta, desde un select hasta un update o insert.

# Codeigniter – URL

## Formato

La URL en codeigniter tiene el siguiente formato:

**/index.php/controlador/accion/**

- Controlador: El nombre de la clase del controlador al que queremos acceder
- Acción: Método dentro del controlador (dentro de la clase)

## URL amigable

Podemos generar reglas en el .htaccess de nuestra aplicación para no tener el archivo index.php en las URLs de nuestra aplicación:

- Debemos mover el archivo .htaccess ubicado en la carpeta application a la raíz del proyecto.
- Debemos copiar las siguientes reglas

```
<IfModule mod_rewrite.c>

    # Suponiendo que el listado de directorios esté desact
    Options +FollowSymLinks -Indexes
    RewriteEngine on

    # Usualmente "AllowOverride" debería estar en 'All' en
    #AllowOverride All

    # Se supone que la regla aplica desde el directorio do
    #RewriteBase /

    # con esta instrucción bloqueamos el acceso a la carpe
    RedirectMatch 403 ^/(system).*$

    # Antes de redireccionar, se verifica que la petición
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d

    # Si el archivo/directorio no existe, redireccionamos
    RewriteRule ^(.*)$ /carrito_compra/index.php/$1 [L]
</IfModule>
```

# Ejercicio de la unidad

Crear una tabla productos en la base de datos que tenga las siguientes columnas:

- Id
- Feliminado
- Falta
- Nombre
- Precio
- Stock

Crear un modelo de productos que tenga métodos para leer un producto dado un id y otro método que retorne todos los productos almacenados en la base de datos.

Crear un controlador que lea los productos de la base de datos y llame a una vista que liste los mismos por pantalla.

# Bibliografía utilizada y sugerida

http://www.desarrolloweb.com/articulos/que-es-mvc.html

http://www.desarrolloweb.com/manuales/manual-codeigniter.html

https://www.codeigniter.com/download

http://fernando-gaitan.com.ar/codeigniter-parte-6-modelos/

# Lo que vimos:

En esta unidad hemos el concepto del patrón Modelo, Vista, Controlador, sus ventajas y sus facilidades de uso.

Luego hemos visto como se aplica dicho patrón en un framework de PHP como codeigniter.

# Lo que viene:

En la próxima unidad continuaremos viendo codeginter, validación de formularios, envíos de emails y más temas.