



# MANUAL FRAMEWORK SEMILLA





## **COLABORADORES:**

- *Licenciado Freddy Peñalver.*
- *Ingeniero Guzbeny Ramirez.*
- *Ingeniero Rafael Gonzalez.*
- *Ingeniero Edgar Silva.*



---

## CONTENIDO

### INTRODUCCIÓN

#### 1. CONCEPTOS BÁSICOS

1.1. ¿Qué es un Framework?.

1.2. Ventajas del Framework

#### 2. JQUERY

2.1. ¿Qué es JQuery?

2.2. Descarga e Inclusión de JQuery

2.3. Selección de Elementos o Selectores de JQuery.

2.4. Filtros de Formulario

2.5. Filtros de los Selectores

2.6. Refinamiento y Filtrado de Selecciones

2.7. Comprobar Selecciones.

2.8. Guardar Selecciones

2.9. Encadenamiento

2.10. Formateo de código encadenado

2.11. Obtenedores (Getters) & Establecedores (Setters)

2.12. Efectos Incorporados en la Biblioteca.

2.13. Eventos

2.14. Métodos AJAX

2.14.1. \$.load()

2.14.2. \$.get()

2.14.3. \$.post().

2.14.4. \$.getJSON()

2.14.5. \$.ajax().

2.15. JQuery en el Framework Semilla.

#### 3. BASE DE DATOS

3.1. Formas Normales

3.2. Estándar del Framework



### 3.3. Ejemplos.

## 4. MVC

### 4.1. Modelo-Vista-Controlador (MVC).

### 4.2. ¿Por qué Utilizar MVC?

### 4.3. Modelo

### 4.4. Vista

### 4.5. Controlador

### 4.6. Ejemplos que demuestran ventajas de MVC

### 4.7. MVC en el Framework Semilla

### 4.8. Diagrama MVC en el Framework Semilla

## 5. POO

### 5.1. ¿Qué es POO?

### 5.2. POO en el Framework Semilla

## 6. INSTALACIÓN

### 6.1. ¿Qué se debe tener?

### 6.2. Primeros Pasos

### 6.3. Generar MVC de Nuevas Tablas

## 7. LIBRERÍAS DEL FRAMEWORK

## 8. PREGUNTAS FRECUENTES

## 9. TIPS PARA EL MANEJO DEL FRAMEWORK SEMILLA



---

## INTRODUCCIÓN

El presente manual nos proporciona las primeras orientaciones que permiten ver que es posible utilizar y entender las nuevas técnicas de programación orientadas a mejorar el rendimiento del programador, permitiendo tener una aplicación rápida en la cantidad de tiempo mínimo disponible, como más adelante podrás apreciar, ya que te aseguro que aprendiendo las herramientas por el cual esta echo este **framework semilla**, lograrás hacer grandes aplicaciones, medias o pequeñas según tu necesidad o la necesidad de la organización.

En este manual podrás apreciar cual es la importancia de la abstracción del código, digamos hacer la mayor separación del mismo, dejando que el poder de cada lenguaje trabaje de manera armónica, para el dominio de tu sistema, como ejemplo previo, te toparas con la librería JQuery, que no es más que una poderosa herramienta que esta soportada en el lenguaje de programación famosamente conocido JavaScript, patrones de diseño como el bien difundido MVC, la importancia de un ORM implementada con POO, así como métodos mágicos nativos de PHP, además algo denominado Layouts.

Todo lo anterior nos dará la certeza, que trabajaremos de forma estándar, o lo más estándar posible, para poder impulsar el crecimiento del conocimiento colectivo de todos, donde tú serás parte importante de todo este gran proyecto de aprendizaje.

Te invito a que tengas la mente abierta a estos nuevos paradigmas de programación, ya que sé que eres programador, y como sabrás en nuestra área de desarrollo profesional las cosas van cambiando, y nosotros tenemos la obligación hasta donde sea posible estar en mejoras continuas, entonces te exhorto a preguntar cuando sea necesario, buscar en internet, estos u otros términos que encuentres en el manual

No quiero dejar pasar la oportunidad para motivarte a suministrar los aportes necesarios, para que esta “*orgánica*” herramienta, siga su crecimiento y desarrollo.

Unos Pensamientos:

- *Si quieres hacer realidad lo imposible, simplemente patea la “im” y conviértelo en*



*“posible”. (pensamiento SCOUT)*

- *Dame una palanca lo suficientemente fuerte y un punto de apoyo... y moveré la tierra.*  
*(Arquimedes)*
- *Las oportunidades se multiplican a la medida que son aprovechadas. (Sun Tzu)*
- *La locura está en pretender obtener un resultado diferente, haciendo siempre lo mismo.*  
*(Albert Einstein)*
- *Si quieres cambiar al mundo, participa promoviendo el cambio (Freddy Peñalver)*



# ***1. CONCEPTOS BÁSICOS***



---

## **1.1. ¿QUÉ ES UN FRAMEWORK?**

Un framework es una herramienta diseñada para apoyar el desarrollo de sitios web dinámicos, aplicaciones web o sistemas de información además de servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto y teniendo como objetivos principales: la rapidez, productividad y profesionalidad de los productos que se obtengan mediante la herramienta.

## **1.2. VENTAJAS DEL FRAMEWORK**

- Reducción en los tiempos de respuestas de 3 meses a 3 minutos.
- Mejor metodología de implementación de patrones de diseño.
- Abstracción de los distintos lenguajes.
- Separación del código.
- Renderización de las vistas (Layouts).





## ***2. JQUERY***



## 2.1. ¿Qué es JQuery?

JQuery es la librería JavaScript que ha irrumpido con más fuerza. Su autor original es John Resig, aunque como sucede con todas las librerías exitosas, actualmente recibe contribuciones de decenas de programadores. jQuery también ha sido programada de forma muy eficiente y su versión comprimida apenas ocupa 20 KB. Su diseño interno tiene algunas conceptos importantes, sobre todo el "encadenamiento" de llamadas a métodos. La documentación de jQuery es muy completa en inglés e incluye muchos ejemplos. Además, también existen algunos recursos útiles en español para aprender su funcionamiento básico: <http://docs.jquery.com/>.

## 2.2. Descarga e Inclusión de JQuery

El JQuery debe de ser cargado en en la etiqueta <head>, de la página web, como de talla el ejemplo:

### INCLUSIÓN DE JQUERY

```
<html>

  <head>

    <script src="jquery.min.js" type="text/javascript"></script>

  </head>

<body></body>

</html>
```



No es posible interactuar de forma segura con el contenido de una página hasta que el documento no se encuentre preparado para su manipulación. jQuery permite detectar dicho estado a través de la declaración `$(document).ready()` de forma tal que el bloque se ejecutará sólo una vez que la página esté disponible.

#### SINTAXIS PARA LA EJECUCIÓN DE JQUERY

```
$(document).ready(  
    function()  
    {  
        console.log('El documento esta preparado');  
        alert('El documento esta preparado');  
    }  
);
```

## 2.3. Selección de Elementos o Selectores de JQuery

El concepto más básico de jQuery es el de seleccionar algunos elementos y realizar acciones con ellos. La biblioteca soporta gran parte de los selectores CSS3, En <http://api.jquery.com/category/selectors/> se puede encontrar una completa referencia sobre los selectores de la biblioteca.










Todo inicia con la función **\$()**:

SELECTOR	DEFINICIÓN	SINTAXIS
Etiquetas	<ul style="list-style-type: none"><li>■ Selecciona todos los enlaces de la página.</li></ul>	<code>\$('a');</code>
Identificador	<ul style="list-style-type: none"><li>■ Selecciona el elemento cuyo id sea "primero"</li></ul>	<code>\$('#primero');</code>
Clase	<ul style="list-style-type: none"><li>■ Selecciona todos los elementos con class "titular".</li><li>■ Selecciona todos los h1 con class "titular".</li></ul>	<code>\$('.titular');</code>  <code>\$(h1.titular');</code>
Concatenar varios selectores distintos	<ul style="list-style-type: none"><li>■ Selecciona todo lo anterior</li></ul>	<code>\$(a, #primero, h1.titular');</code>

Las posibilidades de la función **\$()** van mucho más allá de estos ejemplos sencillos, ya que soporta casi todos los selectores definidos por CSS 3 (algo que dispondrán los navegadores dentro de varios años) y también permite utilizar XPath:

SELECTOR	DEFINICIÓN	SINTAXIS
Párrafos	<ul style="list-style-type: none"><li>■ Selecciona todos los párrafos de la página.</li></ul>	<code>\$("p");</code>
Párrafos con enlaces	<ul style="list-style-type: none"><li>■ Selecciona todos los párrafos de la página que tengan al menos un enlace.</li></ul>	<code>\$(p[a]);</code>



SELECTOR	DEFINICIÓN	SINTAXIS
radiobutton	 Selecciona todos los radiobutton de los formularios de la página.	<code>\$('input:radio');</code>
Filtro de enlaces	 Selecciona todos los enlaces que contengan la palabra "Imprimir".	<code>\$('a:contains("Imprimir")');</code>
div	 Selecciona todos los div.	<code>\$('div');</code>
div visibles	 Selecciona los div que no están ocultos.	<code>\$('div:visible');</code>
Elementos pares de una lista	 Selecciona todos los elementos pares de una lista.	<code>\$('ul#menuPrincipal li:even');</code>
Elementos impares de una lista	 Selecciona todos los elementos impares de una lista.	<code>\$('ul#menuPrincipal li:odd');</code>
Párrafos	 Selecciona los 5 primeros párrafos de la página.	<code>\$('p:lt(5)');</code>

Como se pudo observar, las posibilidades de la función `$()` son prácticamente ilimitadas, por lo que la documentación de *jQuery* sobre los selectores (<http://api.jquery.com/category/selectors/>) disponibles es la mejor forma de descubrir todas sus posibilidades.






## 2.4. Filtros de Formulario


Este tipo de filtros funcionan más o menos igual que los selectores estudiados en el punto 2.3. solo que son específicos para ayudarnos a trabajar con formularios.

FILTRO	DEFINICIÓN	SINTAXIS
Input	■ Encuentra todos los input: select, textarea, button.	<code>\$(':input');</code>
Text	■ Selecciona todos los elementos de texto.	<code>\$(':text');</code>
Password	■ Selecciona todos los elementos tipo password.	<code>\$(":password");</code>
Checkbox	■ Selecciona todos los elementos de tipo checkbox.	<code>\$(':checkbox');</code>
Submit	■ Selecciona todos los elementos de tipo submit.	<code>\$(':submit');</code>
Reset	■ Selecciona todos los elementos de de tipo reset.	<code>\$(':reset');</code>
Image	■ Selecciona todos los elementos de tipo image.	<code>\$(':image');</code>
Button	■ Selecciona todos los elementos button.	<code>\$(":button");</code>
FILTRO	DEFINICIÓN	SINTAXIS



File	 Selecciona todos los elementos file.	<code>\$(":file");</code>
Elementos habilitados	 Selecciona todos los elementos que están habilitados.	<code>\$(":enabled");</code>
Elementos inhabilitados	 Selecciona todos los elementos que están inhabilitados.	<code>\$(":disabled");</code>
Check	 Selecciona todos los elementos que están en estado de "check" como radiobuttons o checkboxes.	<code>\$(":checked");</code>
Elementos seleccionados	 Selecciona todos los elementos que están seleccionados.	<code>\$(":selected");</code>

Ejemplo:

DEFINICIÓN	HTML	JQUERY
 Obtiene todos los elementos input dentro del formulario #mi_form.	<pre>&lt;form id="mi_form" &gt;     &lt;input name="nombre" /&gt;     &lt;input name="apellido" /&gt; &lt;/form&gt;</pre>	<code>\$("#myForm:input");</code>



## 2.5. Filtros de los Selectores

FILTRO	DEFINICIÓN	SINTAXIS
Primer Elemento	■ Obtiene el primer elemento "p".	<code>\$("p:first");</code>
Último Elemento	■ Obtiene el último elemento p	<code>\$("p:last");</code>
Elementos Pares	■ Obtiene los elementos "p" pares	<code>\$("p:even");</code>
Elementos Impares	■ Obtiene los elementos "p" impares	<code>\$("p:odd");</code>
Primer elemento con clase aplicada	■ Obtiene el primer elementos que tenga la clase "a" aplicada	<code>\$(".a:last");</code>
Elementos pares con clase aplicada	■ Obtiene los elementos pares que tengan la clase "b" aplicada	<code>\$(".b:even");</code>
Elementos después de un índice	■ Obtiene todos los elementos que están después del índice 1	<code>\$("p:gt(1)");</code>
	■ Obtiene los elementos "p" menos el que esta en la posición 2	<code>\$("p:not(p:eq(2))");</code>





## 2.6. Refinamiento y Filtrado de Selecciones

A veces, se puede obtener una selección que contiene más de lo que necesita; en este caso, es necesario refinar dicha selección, *jQuery* ofrece varios métodos para poder obtener exactamente lo que desea.

DEFINICIÓN	SINTAXIS
El elemento <code>div.foo</code> contiene elementos <code>&lt;p&gt;</code>	<code>\$('.div.foo').has('p');</code>
El elemento <code>h1</code> no posee la clase <code>'bar'</code>	<code>\$('.h1').not('.bar');</code>
Un ítem de una lista desordenada que posee la clase <code>'current'</code>	<code>\$('.ul li').filter('.current');</code>
El primer ítem de una lista desordenada	<code>\$('.ul li').first();</code>
El sexto ítem de una lista desordenada	<code>\$('.ul li').eq(5);</code>

## 2.7. Comprobar Selecciones

Para determinar si el contenido de la selección tiene elementos se utiliza la propiedad `length`:

SINTAXIS JQUERY
<code>if (\$('div.foo').length) { ... }</code>



## 2.8. Guardar Selecciones

En el siguiente ejemplo “Guardar selecciones en una variable”, la variable comienza con el signo de dólar. Contrariamente a otros lenguajes de programación, en JavaScript este signo no posee ningún significado especial — es solamente otro carácter. Sin embargo aquí se utilizará para indicar que dicha variable posee un objeto jQuery.

SINTAXIS JQUERY
<code>var \$divs = \$('div');</code>

## 2.9. Encadenamiento

Si en una selección se realiza una llamada a un método, y éste devuelve un objeto *jQuery*, es posible seguir un “encadenado” de métodos en el objeto.

HTML	JQUERY	RESULTADO
<code>&lt;div id="div_h3"&gt;</code>	<code>\$('#div_h3')</code>	PRIMER H3
<code>    &lt;h3&gt;PRIMER H3&lt;/h3&gt;</code>	<code>.find('h3')</code>	
<code>    &lt;h3&gt;SEGUNDO H3&lt;/h3&gt;</code>	<code>.eq(2)</code>	SEGUNDO H3
<code>    &lt;h3&gt;TERCER H3&lt;/h3&gt;</code>	<code>.html('MODIFICADO');</code>	MODIFICADO
<code>    &lt;h3&gt;CUARTO H3&lt;/h3&gt;</code>		CUARTO H3
<code>&lt;/div&gt;</code>		



## 2.10. Formateo de código encadenado

Por otro lado, si se está escribiendo un encadenamiento de métodos que incluyen muchos pasos, es posible escribirlos línea por línea, haciendo que el código luzca más agradable para leer.

SINTAXIS JQUERY NORMAL
<code>\$('#content').find('h3').eq(2).html('nuevo texto para el tercer elemento h3');</code>
SINTAXIS JQUERY FORMATEADA
<pre>\$('#content')     .find('h3')     .eq(2)     .html('nuevo texto para el tercer elemento h3');</pre>

## 2.11. Obtenedores (Getters) & Establecedores (Setters)

jQuery “sobrecarga” sus métodos, es decir, el método para establecer un valor posee el mismo nombre que el método para obtener un valor. Cuando un método es utilizado para establecer un valor, es llamado método establecedor (en inglés setter). En cambio, cuando un método es utilizado para obtener (o leer) un valor, es llamado obtenido (en inglés getter).

- El método utilizado como establecedor:

HTML	JQUERY	RESULTADO
<code>&lt;h1&gt;</code> <i>Valor Inicial del HTML</i> <code>&lt;/h1&gt;</code>	<code>\$('h1').html('Escribo Algo');</code>	Escribo Algo
	<code>\$('h1').text('Escribo un Texto');</code>	Escribo un Texto
<code>&lt;input value='juan'/&gt;</code>	<code>\$('input[name=nom]').val('EDGAR');</code>	<input type="text" value="EDGAR"/>
<code>&lt;input id='id_in' name='a'/&gt;</code>	<code>\$('#id_in').attr('name','b');</code>	<code>&lt;input d='id_in' name='b'&gt;</code>



- El método utilizado como obtenedor:

HTML	JQUERY	RESULTADO
<h1>  Valor Inicial del HTML </h1>	var \$h1= \$('h1').html();	En la variable \$h1 queda guardado 'Valor Inicial del HTML'
	var \$h1=\$('h1').text();	En la variable \$h1 queda guardado 'Valor Inicial del HTML'
<input value='juan' name='a' />	var \$valor=\$('input[name=a]').val();	En la variable \$valor queda guardado 'juan'
<input name="e" id="edgar" />	var \$valor=\$('#edgar').attr('name')	En la variable \$valor queda guardado 'e'

## 2.12. Efectos Incorporados en la Biblioteca

Los efectos más utilizados ya vienen incorporados dentro de la biblioteca en forma de métodos:

DEFINICIÓN	SINTAXIS
Muestra el elemento seleccionado.	<code>\$('#id_elemento').show() ;</code>
Oculto el elemento seleccionado.	<code>\$('#id_elemento').hide()</code>
Oculto o Muestra el elemento seleccionado, según su contrario.	<code>\$('#id_elemento').toggle();</code>
De forma animada, cambia la opacidad del elemento seleccionado al 100%.	<code>\$('#id_elemento').fadeIn() ;</code>
De forma animada, cambia la opacidad del elemento seleccionado al 0.	<code>\$('#id_elemento').fadeOut();</code>



DEFINICIÓN	SINTAXIS
Muestra el elemento seleccionado con un movimiento de deslizamiento vertical.	<code>\$('#id_elemento').slideDown();</code>
Oculto el elemento seleccionado con un movimiento de deslizamiento vertical.	<code>\$('#id_elemento').slideUp();</code>

## 2.13. Eventos

En jQuery hay varias formas de manejar los eventos, entre ellas las funciones `click()`, `focus()`, `blur()`, `change()`, `keypress()`. A partir de jQuery 1.7 se incluye también la función `on()` que reúne todas estas funciones, permitiendo manejar tanto eventos sencillos como casos de propagación de eventos. Ejemplos de eventos:

SINTAXIS	
<code>\$('#miBoton').on(     'click',     function()     {         alert('Clic en el botón');     } );</code>	<code>\$('#miBoton').click(     function()     {         Alert('Clic en el botón');     } );</code>



Ahora veamos un ejemplo con múltiples eventos:

### SINTAXIS

```
$('#MiTabla tr').on(  
    {  
        click: function ()  
        {  
            $(this).toggleClass('over');  
            // Otras acciones  
        },  
        dblclick:function()  
        {  
            $(this).toggleClass('over');  
            // Otras acciones  
        }  
    }  
);
```

Para mayor información podemos utilizar estos extensos:

EXTENSION	DEFINICIÓN
<i>.bind()</i>	Para asignar más de una función a un objeto.
<i>.blur()</i>	Cuando el foco deja de estar en un elemento.
<i>.change()</i>	Valor del elemento cuando cambia.
<i>.click()</i>	El elemento(s) ha sido clickado.
<i>.dblclick()</i>	El elemento(s) ha recibido un doble click.
<i>.die()</i>	Elimina todos los eventos que fueron creados con el método live().
<i>.error()</i>	Para manejo de errores javascript.
<i>.focus()</i>	Cuando el elemento recibe el foco del cursor.



EXTENSION	DEFINICIÓN
<i>.focusin()</i>	Detecta el foco de un campo de entrada anidado dentro del elemento padre. Esta función espera al elemento padre como selección.
<i>focusout()</i>	Como el anterior pero cuando se pierde el foco.
<i>.hover()</i>	Cuando el puntero se encuentra dentro de un elemento. También puede ser controlado con las 2 funciones <i>mouseenter()</i> y <i>mouseleave()</i> .
<i>.keydown()</i>	Cuando una tecla es pulsada.
<i>.keypress()</i>	Como el anterior, para garantizar el correcto comportamiento de los diferentes navegadores.
<i>.keyup()</i>	Cuando se “suelta” una tecla; es decir, cuando deja de ser pulsada.
<i>.live()</i>	Añade eventos a los elementos que cumplen la selección. A partir de ese momento se añadirán las funciones a todos los elementos que coincidan con los criterios. Es útil cuando se van agregando elementos de forma dinámica.
<i>.load()</i>	Lanza una función cuando todos los elementos y sus subelementos han sido cargados. Usado, por ejemplo, para la carga de imágenes.
<i>.mousedown()</i>	El botón del mouse se ha clickado en la selección.
<i>.mouseenter()</i>	El puntero del mouse ha entrado en el elemento.
<i>.mouseleave()</i>	El puntero del mouse sale del elemento.
<i>.mousemove()</i>	El puntero del mouse se mueve sobre la superficie del elemento.
<i>.mouseout()</i>	El mouse sale del elemento.
<i>.mouseover()</i>	El mouse se mueve dentro del propio elemento o de alguno de sus hijos. La diferencia entre <i>mouseenter()</i> es que éste último solo hace referencia al elemento en sí y <i>mouseover()</i> tiene en cuenta a sus descendientes.
<i>mouseup()</i>	Cuando el puntero del ratón está encima del elemento y el botón del mouse está pulsado.
<i>.off()</i>	Elimina los controladores de eventos para el elemento.
<i>on()</i>	Función avanzada de eventos.



EXTENSION	DEFINICIÓN
<code>.one()</code>	Una vez el evento ocurre, el controlador de eventos se elimina mediante <code>unbind()</code> . Es decir, solo se ejecuta una vez la función.
<code>.ready()</code>	Se ejecuta cuando el DOM está listo pero antes del <code>onload</code> .
<code>.resize()</code>	Cuando cambia el tamaño de la pantalla.
<code>.scroll()</code>	Cuando se hace scroll sobre un elemento cuya propiedad CSS <code>overflow</code> es <code>scroll</code> .
<code>.select()</code>	Cuando se hace una selección de texto de un elemento de campo de texto.
<code>.submit()</code>	Cuando se envía un formulario.
<code>.toggle()</code>	Esta función en realidad ejecuta dos funciones. Primero una, y cuando ésta en modo ON y se vuelve a realizar el mismo evento, se ejecuta la segunda función y la primera se pone en OFF. Sería análogo a un switch.
<code>.unbind()</code>	Elimina un evento de la selección.

## 2.14. Métodos AJAX

El método XMLHttpRequest (XHR) permite a los navegadores comunicarse con el servidor sin la necesidad de recargar la página. Este método, también conocido como Ajax (Asynchronous JavaScript and XML), permite la creación de aplicaciones ricas en interactividad.

Las peticiones Ajax son ejecutadas por el código JavaScript, el cual envía una petición a una URL y cuando recibe una respuesta, una función de devolución puede ser ejecutada la cual recibe como argumento la respuesta del servidor y realiza algo con ella. Debido a que la respuesta es asíncrona, el resto del código de la aplicación continua ejecutándose, por lo cual, es imperativo que una función de devolución sea ejecutada para manejar la respuesta. A través de varios métodos, jQuery provee soporte para Ajax, permitiendo abstraer las





diferencias que pueden existir entre navegadores. Los métodos en cuestión son:

- \$.get().
- \$.getScript().
- \$.getJSON().
- \$.post().
- \$.load().

A pesar que la definición de Ajax posee la palabra “XML”, la mayoría de las aplicaciones no utilizan dicho formato para el transporte de datos, sino que en su lugar se utiliza HTML plano o información en formato JSON (JavaScript Object Notation).

En general, Ajax no trabaja a través de dominios diferentes. Sin embargo, existen excepciones, como los servicios que proveen información en formato JSONP (JSON with Padding), los cuales permiten una funcionalidad limitada a través de diferentes dominios.

MÉTODO	DEFINICIÓN	SINTAXIS
<b>\$.load()</b>	<ul style="list-style-type: none"><li>■ La función \$.load() inserta el contenido de la respuesta del servidor en el elemento de la página que se indica.</li></ul>	<pre>&lt;div id="info"&gt;&lt;/div&gt; \$('#info').load('/ruta/hasta/pagina.php');</pre>
<b>\$.get()</b>	<ul style="list-style-type: none"><li>■ URL: String</li><li>■ data: string u Objeto Json</li><li>■ function: función definida o anónima</li><li>■ dataType: text, xml, json, jsonp, html, script,</li></ul>	<pre>\$.get(URL,data,function(data,status),data Type)</pre>



MÉTODO	DEFINICIÓN	SINTAXIS
<b>\$.get()</b>	<ul style="list-style-type: none"><li>■ Petición GET simple.</li></ul>	<code>\$.get('/ruta/hasta/pagina.php');</code>
<b>\$.get()</b>	<ul style="list-style-type: none"><li>■ Petición GET con envío de parámetros y función que procesa la respuesta.</li></ul>	<code>\$.get( '/ruta/hasta/pagina.php',   {articulo:'34'},   function(datos)   { alert('Respuesta = '+datos);},   "json");</code>
<b>\$.post()</b>	<ul style="list-style-type: none"><li>■ URL: String.</li><li>■ data: string u Objeto Json</li><li>■ function: función definida o anónima</li><li>■ dataType: text, xml, json, jsonp, html, script.</li></ul>	<code>\$.post(URL,data,function(data,status),   dataType);</code>
<b>\$.post()</b>	<ul style="list-style-type: none"><li>■ Petición POST con envío de parámetros y función que procesa la respuesta.</li></ul>	<code>\$.post('/ruta/hasta/pagina.php',   {articulo:'34'},   function(datos){      alert('Respuesta='+datos);   },"json");</code>
<b>\$.getJSON()</b>	<ul style="list-style-type: none"><li>■ URL: String.</li><li>■ data: string u Objeto Json.</li><li>■ function: función definida o anónima.</li></ul>	<code>\$.getJSON(URL,data,function(data,sta tus));</code>



MÉTODO	DEFINICIÓN	SINTAXIS
<b>\$.ajax()</b>	<ul style="list-style-type: none"><li>■ url: para la petición.</li><li>■ data: la información a enviar, también es posible utilizar una cadena de datos.</li><li>■ type: POST o GET.</li><li>■ dataType: el tipo de información que se espera de respuesta.</li><li>■ succes: código a ejecutar si la petición es satisfactoria; la respuesta es pasada como argumento a la función.</li><li>■ error: código a ejecutar si la petición falla; son pasados como argumentos a la función el objeto jqXHR (extensión de XMLHttpRequest), un texto con el estatus de la petición y un texto con la descripción del error que haya dado el servidor.</li><li>■ <i>complete: código a ejecutar sin importar si la petición falló o no.</i></li></ul>	<pre>\$.ajax({     url : 'post.php',     data: {id: 123},     type: 'GET',     dataType: 'json',     <b><u>success</u></b>: function(json){         \$         ('&lt;h1/&gt;').text(json.title).appendTo('body &lt;/h1&gt;');         \$('&lt;div class="content"/&gt;')         .html(json.html).appendT         o('body');     },     <b><u>error</u></b>: function(jqXHR, status,     error){         alert('Disculpe,         existió un problema');     },     <b><u>complete</u></b>: function(jqXHR,     status) {         alert('Petición         realizada');     } });</pre>



## 2.15. Jquery en el Framework Semilla

En el framework semilla la utilización de código jquery es esencial, ya que se utiliza básicamente para las validaciones necesarias de un formulario. Como se puede apreciar en el diagrama de MVC dentro del Framework semilla, la capa de programación en Jquery se encuentra entre la Vista y el Controlador, permitiendo al programador realizar cualquier tipo de validación que esté referida a campos del formulario (acciones al introducir una fecha, consulta al introducir un número o código asociado a un registro, etc).

Esta capa también es la encargada de realizar el envío de los datos del formulario hasta el controlador permitiendo así la continuidad del MVC. Esta comunicación o transferencia de datos se lleva a cabo por medio de un plugin de Jquery llamado Validate.

### Ubicando Nuestros archivos Jquery

En el framework semilla para acceder a los archivos jquery, debemos entrar la carpeta Js de nuestro sistema, la misma contendrá una carpeta validaciones, dentro de la cual se encuentran los archivos js que están relacionados a cada uno de nuestros formularios, agrupados por los diferentes schemas que contenga el sistema. Apariencia de un archivo JS

```
$(document).ready(function() {
/*CODIGO_PROCESAR*/

    if ($("#accion").val()!="buscar") {
        var RULES = {
            usuario:{required: true },
            tipo_factura:{required: true},
        };
    }
    else
    {
        var RULES = {};
    }

    //AREA PARA VALIDACIONES DEL FORMULARIO

    $("#frm_formulario").validate({
        rules: RULES,
        submitHandler: function(form) {
            if ($("#accion").val()=="eliminar" && !(confirm("confirme que quiere eliminar el registro?"))){
                return false;
            }

            $("#div_respuesta").html("<img src='../imagenes/loader.white.gif' />Cargando...");
            dataString = $("#frm_formulario").serialize();
            $.post("../controlador/schema_sigbiys/tbl_tipos_facturas.php", dataString, procesar, "json");
            return false;
        }
    });

    //AREA PARA FUNCIONES DEL FORMULARIO
```



En la imagen se puede apreciar como están compuestos los archivos .js. El mismo contiene dos áreas específicas para comenzar a generar el código de validaciones. Se recomienda respetar las áreas para evitar problemas al momento de revisar nuestro código y de esta forma se garantiza que la aplicación funcionará perfectamente.

La primera área es para el código que esté referido a eventos jquery (onclick, onchange, etc) y la segunda área es para generar funciones que podrán ser usadas múltiples veces dentro del mismo formulario (Funciones que sólo aplican para el formulario en particular)

El código que genera el framework en estos archivos es para garantizar el correcto funcionamiento del plugin validate y por consiguiente de la aplicación. No se debe modificar sino esta seguro de su funcionamiento.

- **Plugin Validate:** Este plugin jQuery hace la validación de formularios más fácil y simple, mientras que ofrece un montón de opciones de personalización. Permite validar por tipo de datos como datos simples, correos, password, confirmación de password y mucho mas. Para familiarizarse con el plugin validate es importante conocer los siguientes conceptos del mismo.
- **Método:** Un método de validación implementa la lógica para validar un elemento, como por ejemplo: un método de correo electrónico, que comprueba el formato correcto del valor de una entrada de texto. El plugin posee una serie de métodos estándares disponibles.
- **Regla:** Una regla de validación asocia un elemento con un método de validación, ejemplo: validar la entrada cuyo nombre es "name:primary-mai" con el método: "required" y "email".



## Métodos para la validación

MÉTODO	DESCRIPCIÓN
required	Hace que el elemento requerido.
remote	Pide un recurso para comprobar el elemento de validez.
minlength	Hace que el elemento requiere una longitud mínima determinada.
maxlength	Hace que el elemento requiere una longitud máxima dado.
rangelength	Hace que el elemento requiere de un rango de valores determinado.
min	Hace que el elemento requieren un mínimo determinado.
max	Hace que el elemento requiere un máximo determinado.
range	Hace que el elemento requiere un rango de valor determinado.
email	Hace que el elemento requiere una dirección de correo electrónico válida
url	Hace que el elemento requiere una URL válida
date	Hace que el elemento requiere una fecha.
dateISO	Hace que el elemento requiere una fecha ISO.
number	Hace que el elemento requiere un número decimal.
digits	Hace que el elemento requiere sólo dígitos.
creditcard	Hace que el elemento requiere un número de tarjeta de crédito.
equalTo	Requiere el elemento a ser el mismo que otro

## Estableciendo Métodos y Reglas con Plugin Validate

En los archivos js creados por el framework en la parte superior tenemos lo siguiente:

En este ejemplo se observa los campos con nombre usuario y tipo\_factura del formulario tienen un método de validación establecido como requerido.



```
if ($("#accion").val()!="buscar") {  
    var RULES = {  
        usuario:{required: true },  
        tipo_factura:{required: true},  
    };  
}  
else  
{  
    var RULES = {};  
}
```

La *var RULES* es una variable que guarda las reglas con las cuales se validará el formulario, esta variable puede alojar todas las reglas necesarias para cumplir con las necesidades de validación de nuestro sistema y se pueden utilizar los métodos necesarios para cada uno de los elementos que intervengan en el formulario. Ejemplo:

```
var RULES = {  
    correo:{  
        required: true,  
        email:true},  
    tipo_factura:{  
        required: true,  
        digits:true},  
};
```

## Validando Formularios

Todos los formularios tienen un boton de accion de tipo *submit* con el cual se indica que el formulario está listo para ser procesado, además los formularios poseen un identificador (id="frm\_formulario"), con el cual entra en funcionamiento el siguiente código:



```
$("#frm_formulario").validate({
  rules: RULES,
  submitHandler: function(form) {

    if (($("#accion").val()=="eliminar") && !(confirm("confirme que quiere eliminar el registro?"))){
      return false;
    }

    $("#div_respuesta").html("<img src='../imagenes/loader.white.gif' />Cargando...");
    dataString = $("#frm_formulario").serialize();
    $.post("../controlador/schema_sigblys/tbl_tipos_facturas.php", dataString, procesar, "json");
    return false;
  }
});
```

Donde se utiliza la variable RULES generada al comienzo del archivo para verificar que se cumplan las reglas establecidas, este proceso se realiza de manera transparente para el programador. En el caso que las reglas se cumplan se procede a realizar el envío de los datos al controlador determinado, mediante .serializer() el cual se encarga generar una variable (dataString) que contendrá todos los datos del formulario para luego ser envia.

El envío se realiza mediante un método \$.post(), la respuesta a este envío de datos es procesada por una función procesar que es genérica para todas las respuestas y se encuentra en el archivo configuracion.js. De ser necesario se puede generar una función procesar para formulario en particular. Un ejemplo de ello sería:

```
$("#frm_formulario").validate({
  rules: RULES,
  submitHandler: function(form) {

    if (($("#accion").val()=="eliminar") && !(confirm("confirme que quiere eliminar el registro?"))){
      return false;
    }

    $("#div_respuesta").html("<img src='../imagenes/loader.white.gif' />Cargando...");
    dataString = $("#frm_formulario").serialize();
    $.post("../controlador/schema_sigblys/tbl_tipos_facturas.php", dataString,
    function procesar(data){
      if(data.estado=="insertado"){
      if(data.estado=="eliminado"){
      if(data.estado=="encontrado"){
      if(data.estado=="actualizado"){
    },
    "json");
    return false;
  }
});
```





---

Se sustituye procesar: que es el nombre de la función que procesa la respuesta de forma automática por una función procesar particular, donde especifica todos los casos de respuesta que se encuentran por defecto.

## **Programar en JQuery**

Para realizar cualquier tipo de programación de código jquery se debe utilizar el área especificada para ello en los archivos .js

Se puede incluir cualquiera de los eventos de jquery y relacionarlo a los elementos del formulario.

Si se quiere realizar funciones genéricas en lenguaje jquery que se puedan utilizar en todos los archivos .js, esta debe estar incluida en el archivo configuracion.js



## ***3. Base de Datos***



## 3.1. Formas Normales

Para construir un sistema de información que responda a un problema real concreto, lo primero que hay que hacer es decidir cuál es el esquema relacional más adecuado. Encontrar la mejor manera de agrupar los datos en forma de tablas y de relacionar éstas entre sí, es la esencia del diseño de bases de datos relacionales. Para ello existen propiedades exigibles llamadas *formas normales* que rigen las condiciones que deben cumplir las tablas.

A continuación se explica las tres formas normales:

- Consideremos la siguiente factura:

**International Widgets**  
742 Evergreen Terrace  
Springfield, MO

**INVOICE**

INVOICE NO: 125  
DATE: September 13, 2002

To: Foo, Inc.  
23 Main St.  
Thorpleburg, TX

Customer No. 56

QUANTITY	ITEM ID	DESCRIPTION	UNIT PRICE	AMOUNT
4	563	56" Blue Freen	3.50	\$14.00
32	851	Spline End (Xtra Large)	.25	\$8.00
5	692	3" Red Freen	12.00	\$60.00
<b>TOTAL DUE</b>				<b>\$82.00</b>

**INVOICE**

INVOICE NO: 126  
September 14, 2002

Customer No. 2

QUANTITY	ITEM ID	DESCRIPTION	UNIT PRICE	AMOUNT
750	692	3" Red Freen	12.00	\$9,000.00
<b>TOTAL DUE</b>				<b>\$10,750.00</b>

FIGURA 1



- Ahora llevemos los datos reflejados de la factura a una hoja excel.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Invoice No.	Date	Cust. No.	Cust. Name	Cust. Address	Cust. City	Cust. State	Item ID	Item Description	Item Qty.	Item Price	Item Total	Order Total Price
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563	56" Blue Fre	4	\$ 3.50	\$ 14.00	\$ 82.00
3								851	Spline End i	32	\$ 0.25	\$ 8.00	\$ 82.00
4								652	3" Red Free	5	\$ 12.00	\$ 60.00	\$ 82.00
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenue	Washington	DC	563	56" Blue Fre	500	\$ 3.50	\$ 1,750.00	\$ 10,750.00
6								652	3" Red Free	750	\$ 12.00	\$ 9,000.00	\$ 10,750.00

FIGURA 1.1

- Si visualizamos la *figura 1.1* se refleja la compra que va haciendo cada consumidor, esto resulta tedioso al momento de realizar ciertas consultas como:

*¿Cuántos "3" Red Freens" ordenó Freens R Us en el 2002?*

*¿Cuáles han sido las ventas totales de 56" Blue Freens en el estado de Texas?*

*¿Qué ítems fueron vendidos en Julio del 2003?*

En la medida que la cuadrícula crece, se va complicando el hecho de encontrar estas respuestas. Al tratar de organizar los datos de forma que podamos, razonablemente, encontrar las respuestas a estas preguntas, estamos comenzando a realizar el proceso de normalización.



## Primera Forma Normal (PFN).

*“No Elementos Repetidos o Grupo de Elementos”*

- Visualicemos las filas 2,3 y 4 de las cuadrículas de la *figura 1.1*, representan toda la información de la factura 125.
- La primera forma normal nos indica que debemos deshacernos de los elementos repetidos, veamos cuáles son: las casillas H2, H3 y H4 contienen una lista de ítems de los ID Numbers (números de identificación). Esto es una columna dentro de la primera fila de nuestra base de datos (BD). De la misma forma, I2–I4 constituyen una simple columna; lo mismo con J2–J4, K2–K4, L2–L4 y M2–M4. Las columnas de la BD son nombradas algunas veces como atributos (las filas y las columnas son tuplas y atributos).
- Se podrá notar que cada una de estas columnas contienen una lista de valores. La primera forma normal anhela la atomicidad: la indivisibilidad de un atributo dentro de partes similares.
- Por lo tanto, queda claro que tenemos que hacer algo con los datos repetidos de ítems de información dentro de la fila para la factura #125. En la *figura 1.1* son las celdas:  
H2 hasta M2  
H3 hasta M3  
H4 hasta M4
- Datos similares se repiten dentro de la fila de la factura #125. Podemos satisfacer la necesidad de la atomicidad de la primera forma normal simplemente separando cada ítem de estas listas en su propia fila.

orders.xls													
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Invoice No.	Date	Cust. No.	Cust. Name	Cust. Address	Cust. City	Cust. State	Item ID	Item Descr	Item Qty.	Item Price	Item Total	Order Total Price
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563 56" Blue Fre	4	\$ 3.50	\$ 14.00	\$ 82.00	
3	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	851 Spline End	32	\$ 0.25	\$ 8.00	\$ 82.00	
4	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	652 3" Red Free	5	\$ 12.00	\$ 60.00	\$ 82.00	
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenue	Washington	DC	563 56" Blue Fre	500	\$ 3.50	\$ 1,750.00	\$ 10,750.00	
6	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenue	Washington	DC	652 3" Red Free	750	\$ 12.00	\$ 9,000.00	\$ 10,750.00	

**FIGURA 1.2.**



- Si analizamos la figura 1.2. nos damos cuenta que en vez de reducir las duplicaciones se introducen datos del cliente, éste problema será resuelto en la explicación de la tercera forma normal.
- La primera forma normal aborda dos cuestiones:  
*Una fila de dato no puede contener grupos repetidos de datos similares (atomicidad) .*  
*Cada fila tiene que tener un único identificador o llave primaria.*
- Procederemos analizar en la *figura 1.2* cuál de las columnas podemos identificar como llave primaria. Un valor que de forma única identifique una fila se llama *llave primaria*; cuando este valor está formado por dos o más columnas, lo llamamos *llave primaria concatenada*.
- Las dos columnas juntas que de forma única identifican a cada fila son *order\_id* y *item\_id*: no hay otras dos columnas que tengan la misma combinación que *order\_id* y *item\_id*. Por lo tanto, pueden ser usados como llave primaria de la tabla. Aunque están en dos columnas diferentes de la tabla, son tratados como una sola entidad. La llamaremos llave primaria concatenada.

order_id	order_date	customer_id	customer_name	customer_address	customer_city	customer_state	item_id	item_description	item_qty	item_price	item_total_price	order_total_price
125	9/13/2002	56	Foo, Inc.	23 Main St., Thi	Thorpeburg	TX	563 56	* Blue Freen	4	\$3.50	\$14.00	\$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Thi	Thorpeburg	TX	851	Soline End (Xtra	32	\$0.25	\$8.00	\$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Thi	Thorpeburg	TX	662 3	Red Freen	5	\$12.00	\$60.00	\$82.00
126	9/14/2002	2	Freens R Us	1600 Pennsylv	Washington	DC	563 56	* Blue Freen	500	\$3.50	\$1,750.00	\$10,750.00
126	9/14/2002	2	Freens R Us	1600 Pennsylv	Washington	DC	662 3	Red Freen	750	\$12.00	\$9,000.00	\$10,750.00

Record: 14 of 6

**Figura 1.3**

- Las columnas señaladas serán nuestra llave primaria y las podemos denotar con las siglas PK.



- Hasta los momentos nuestro esquema de base datos satisface los dos requerimientos de la primera forma normal: atomicidad y unicidad.

orders
order_id (PK)
order_date
customer_id
customer_name
customer_address
customer_city
customer_state
item_id (PK)
item_description
item_qty
item_price
item_total_price
order_total_price

**FIGURA 1.4.**

## Segunda Forma Normal (SFN).

*“Sin Dependencias Parciales en Llaves Concatenadas”*

Sin dependencias parciales sobre la llave concatenada significa que para una tabla que tiene una llave primaria concatenada, cada columna de la tabla, que no forma parte de la llave primaria, tiene que depender de la llave concatenada completamente. Si hay alguna columna que solamente dependa de una parte de la llave concatenada, entonces decimos que la tabla completa no cumple la Segunda Forma Normal (SFN) y tenemos que crear otra tabla para rectificar este fallo.

Para entender esto, tomemos la tabla de órdenes (figura 1.4) columna a columna y para cada una hagámonos la pregunta:

*¿Puede esta columna existir sin una de las partes de la llave primaria concatenada?*

Si la respuesta es “sí” – aunque sea una vez – entonces la tabla falló a la SFN.


La estructura de la tabla orders de la *figura 1.4*:




*order\_id*: identifica la factura de donde proviene este ítem.


*item\_id*: es el identificador único del renglón del inventario.

No analizamos estas columnas (ya que ellas son parte de la llave primaria). Ahora consideraremos las restantes columnas:

■ *order\_date*: es la fecha en que fue hecha la orden. Es obvio que depende de *order\_id*; la fecha de la orden tiene que tener una orden si no sería solo una fecha. ¿Pero puede una fecha de orden existir sin un *item\_id*?, la respuesta inmediata es si: *order\_date* depende de *order\_id* pero no de *item\_id*. Por lo tanto: *order\_date* no aprobó la SFN. 

De esta forma, nuestra tabla no ha aprobado la SFN. Continuemos probando las otras columnas. Debemos encontrar todas las columnas que no aprueben la prueba para, entonces, hacer algo especial con ellas.

■ *customer\_id*: es el número de identificación del consumidor que puso la orden. ¿Depende el de *order\_id*? No: un consumidor puede existir sin solicitar ninguna orden. Depende el de *item\_id*? No: por la misma razón. Esto es interesante: *customer\_id* (junto con las otras columnas *customer\_\**) no depende de ninguno de los miembros de la llave primaria. ¿Que hacemos con estas columnas? No debemos preocuparnos por ellas hasta que lleguemos a la tercera forma normal. Por ahora, las marcamos como desconocidas. 

■ *Item\_description*: es la próxima columna que no es por si misma parte de la llave primaria. Esta es la descripción del ítem que está en el inventario. Esta columna depende de *item\_id*. ¿Pero puede existir sin una *order\_id*? Si! Un ítem del inventario (junto con su descripción) puede estar en el almacén por un largo tiempo y nunca ser vendido independientemente de la orden. *Item\_description* falló la prueba. 





- **Item\_qty:** se refiere al número de ítems comprados en una factura en particular. ¿Puede la cantidad existir sin el `item_id` ? Imposible: no podemos hablar de la “cantidad de nada” (por lo menos en el diseño de BD). ¿Puede una cantidad existir sin una orden? No: una cantidad que es comprada por medio de una factura no tiene sentido sin la factura. De esta forma, esta columna no viola la SFN: `item_qty` depende de las dos partes de la llave primaria concatenada. ✓
- **Item\_price:** es similar a `item_description`. Depende de `item_id` pero no de `order_id`, de aquí que viola la SFN. ✗
- **Item\_total\_price:** es un caso difícil. De un lado, parece que depende de ambas `order_id` y `item_id` , en cuyo caso pasa la SFN. Por otro lado, es un valor derivado de otros: es el producto de `item_qty` y `item_price`. ¿Que hacer con este campo? De hecho, este campo no pertenece a nuestra BD. El puede ser fácilmente calculado e incluirlo en la BD sería redundante (y fácilmente podría introducir problemas). Por tanto, lo descartaremos y no hablaremos mas de él.
- **order\_total\_price:** es la suma de todos los campos `item_total_price` para una orden en particular, es otro valor derivado de otros valores por tanto, lo descartamos.

He aquí el resultado del análisis de la tabla `orders` (figura 1.4) para la SFN:

FIGURA 1.5

orders	
<code>order_id</code> (PK)	
<code>order_date</code>	✗
<code>customer_id</code>	?
<code>customer_name</code>	?
<code>customer_address</code>	?
<code>customer_city</code>	?
<code>customer_state</code>	?
<code>item_id</code> (PK)	
<code>item_description</code>	✗
<code>item_qty</code>	✓
<code>item_price</code>	✗
<del><code>item_total_price</code></del>	
<del><code>order_total_price</code></del>	



¿Que hacemos con una tabla que falla la SFN, como esta que analizamos? Primero tomamos la segunda mitad de la llave primaria concatenada (ítem\_id) y la ponemos en su propia tabla.

Todas las columnas que dependen de ítem\_id, ya sea completamente o parcialmente, irán con ella a una nueva tabla. Llamaremos a esta nueva tabla order\_items (ver Figura D). Los otros campos que dependen de la primera parte de la llave primaria (order\_id) y los que no estamos seguros, se quedan donde están.

The screenshot shows a database application interface with two tables displayed. The top table is 'orders' and the bottom table is 'order\_items'.

**orders : Table**

order_id	order_date	customer_id	customer_name	customer_address	customer_city	customer_state
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC

Record: 3 of 3

**order\_items : Table**

order_id	item_id	item_description	item_qty	item_price
125	563	56" Blue Freen	4	\$3.50
125	851	Spline End (Xtra	32	\$0.25
125	652	3" Red Freen	5	\$12.00
126	563	56" Blue Freen	500	\$3.50
126	652	3" Red Freen	750	\$12.00

Record: 1 of 5

FIGURA 1.6



Hay varias cosas que destacar:

- Hemos traído una copia de la columna `order_id` de la tabla `order_items`. Esto permite a cada `order_item` “recordar” a cual orden pertenece.
- La tabla `orders` tiene menos columnas que antes.
- La tabla `orders` ya no tiene una llave primaria concatenada. La llave primaria consiste ahora en una sola columna, `order_id`.
- La tabla `order_items` si tiene una llave primaria concatenada.

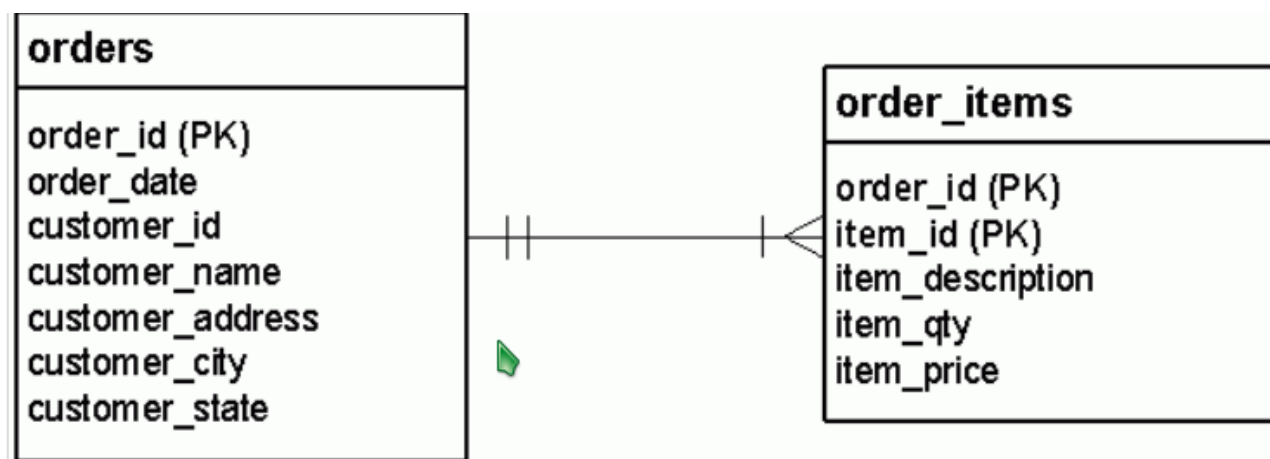
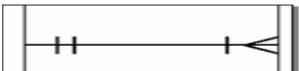


FIGURA 1.7

La línea  significa:

- Cada orden puede ser asociada con cualquier número de `order_items`, y por lo menos uno.
- Cada `order_item` está asociado con una orden y solo una.

Ahora bien nuestra nueva tabla `order_item` sin embargo, aún tiene una llave primaria concatenada. Tenemos que pasarla una vez más por el análisis de la SFN y ver si es capaz de pasarlo. Hacemos la misma pregunta que hicimos antes:

*¿Puede esta columna existir sin una o la otra parte de la llave primaria concatenada?*

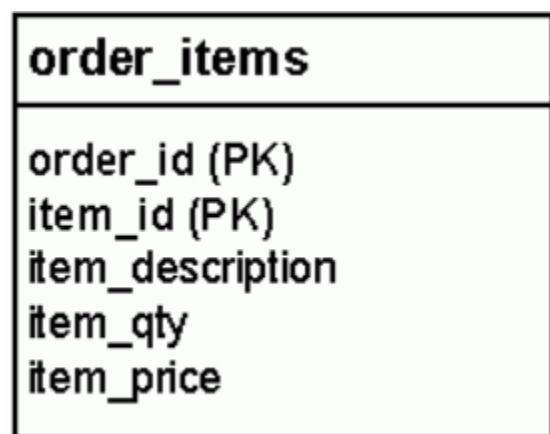





FIGURA 1.8

Ahora considere las columnas que no son parte de la llave primaria:

- `item_description` :depende de `item_id`, pero no de `order_id`. De manera que (sorpresa), esta columna, una vez mas falla la SFN. 
- `item_qty`: depende de ambos miembros de la llave primaria, por lo que no viola la SFN. 
- `item_price`: depende de `item_id` pero no de `order_id`, de manera que si viola la SFN. 

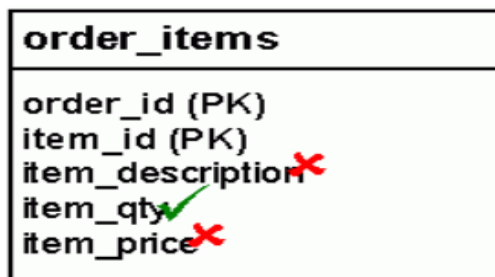


FIGURA 1.9



De esta forma, con los campos que fallaron la SFN, creamos una nueva tabla que llamaremos items.

order_id	item_id	item_qty
125	563	4
125	851	32
125	652	5
126	563	500
126	652	750

item_id	item_description	item_price
563	56" Blue Freen	\$3.50
851	Spline End (Xtra Large)	\$0.25
652	3" Red Freen	\$12.00

FIGURA 1.10

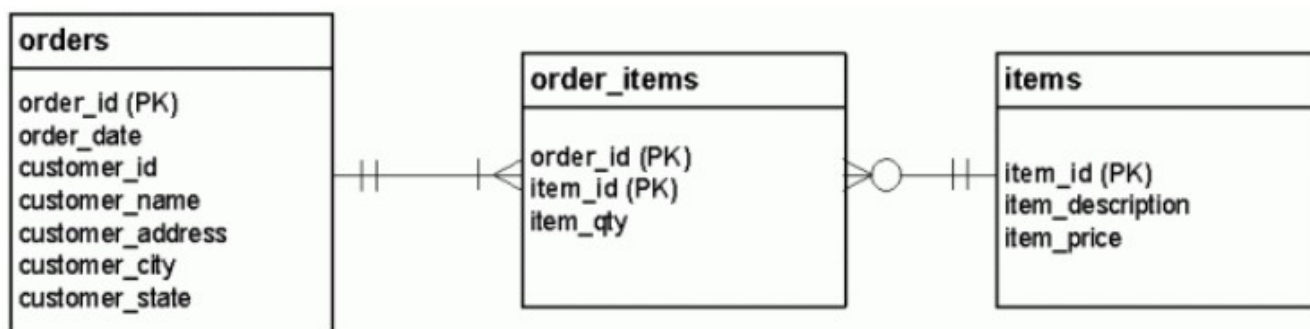


FIGURA 1.11

La línea que conecta las tablas items y order\_items significa lo siguiente: Cada item puede ser asociado con cualquier número de líneas de cualquier número de factura, incluso cero; cada order-item está asociado con un item, y solo uno. Estas dos líneas son ejemplos de relaciones de uno a muchos.



### Tercera Forma Normal (TFN).

*“Sin dependencia de atributos que no son llaves”.*

Como está nuestra BD ahora, si un consumidor hace mas de una orden, tenemos que introducir todas esas informaciones de contactos de los consumidores una vez mas. Eso sucede porque hay columnas en la tabla orders que dependen de “atributos que no son llaves”.

Para entender mejor este concepto, consideremos:

- order\_date. ¿Puede ella existir independientemente de la columna order\_id? No: una fecha de orden no tiene sentido sin una orden. order\_date digamos que depende de un atributo llave (order\_id es el “atributo llave” porque es la llave primaria de la tabla).
- customer\_name: ¿Puede existir por si solo, fuera de la tabla orders? Si: Es completamente razonable hablar de un consumidor sin necesidad de hablar de ordenes o facturas. Lo mismo pasa con customer\_address, customer\_city y customer\_state.

Estas cuatro columnas actualmente dependen de customer\_id, que no es una llave en esta tabla (es un atributo que no es llave). Estos campos pertenecen a su propia tabla, con customer\_id como llave primaria.

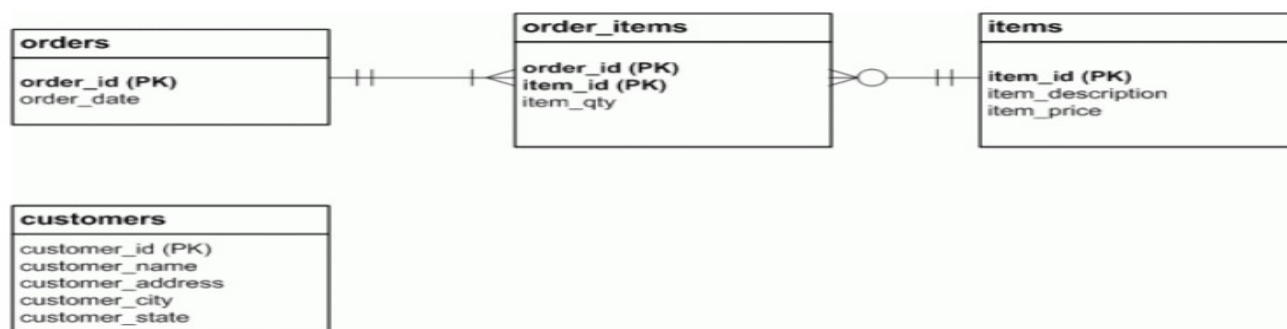
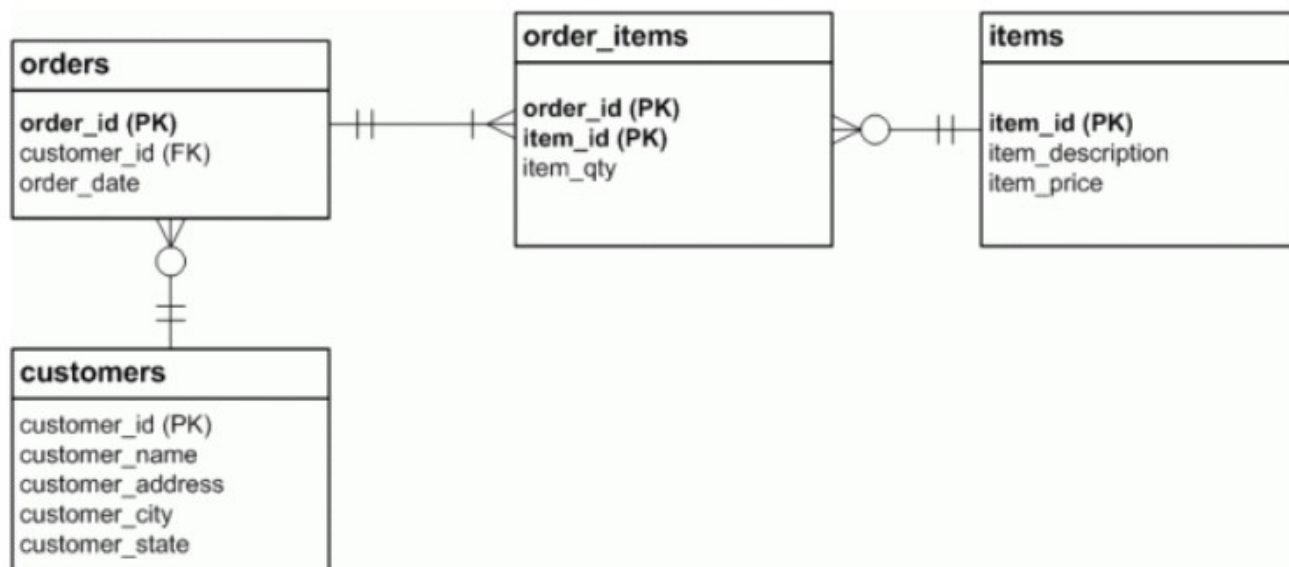


FIGURA 1.12



Se puede observar que se ha cortado la relación entre la tabla orders y los datos del Consumidor que antes estaban en ella.

Tenemos que restaurar la relación creando una entidad llamada llave externa (indicada en nuestro diagrama como (FK) en la tabla orders. Una llave externa es, en esencia, una columna que apunta a la llave primaria en otra tabla.



**FIGURA 1.13**

La relación que ha sido establecida entre las tablas orders y customers puede ser expresada de la siguiente manera: cada orden es hecha por un, y solo un consumidor; cada consumidor puede hacer cualquier número de órdenes, incluso cero. Una puntualización final... Notará que las columnas order\_id y item\_id en order\_items cumplen una doble función: no solamente como llave primaria (concatenada), sino que también, individualmente, sirven como llaves externas de las tablas orders y ítems respectivamente.

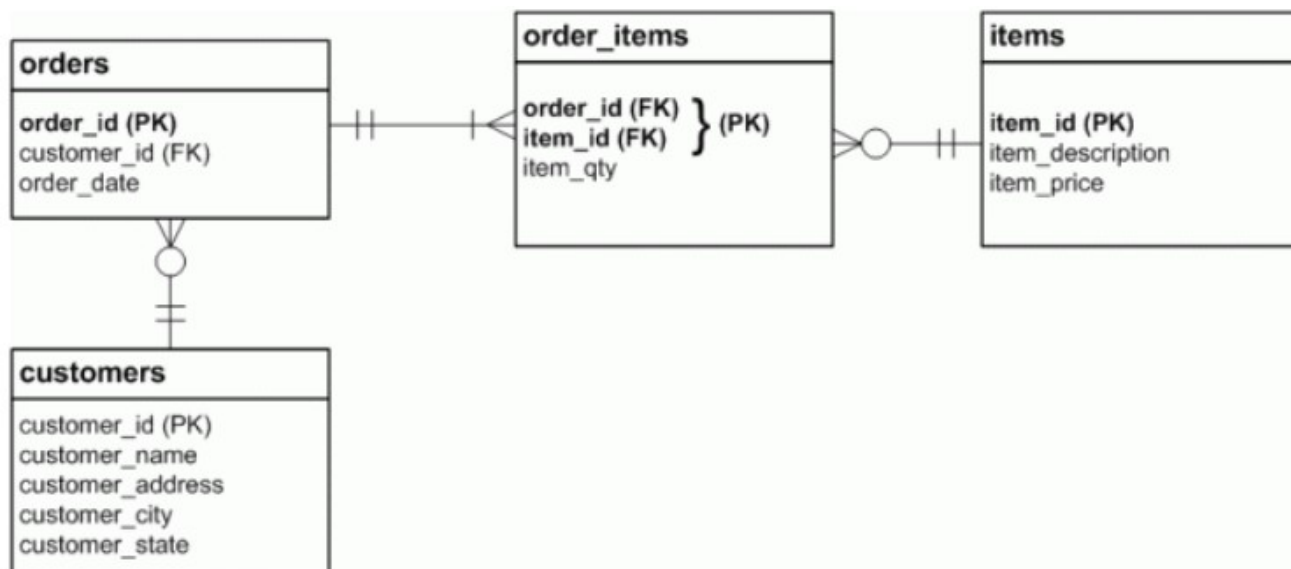


FIGURA 1.14

Y finalmente, vemos como quedan los datos en cada una de las cuatro tablas.

The screenshot displays four database tables with their respective data:

order_id	customer_id	order_date
125	56	9/13/2002
126	2	9/14/2002

order_id	item_id	item_qty
125	563	4
125	851	32
125	652	5
126	563	500
126	652	750

customer_id	customer_name	customer_address	customer_city	customer_state
56	Foo, Inc.	23 Main St., Thi	Thorpeburg	TX
2	Freens R Us	1600 Pennsylv	Washington	DC

item_id	item_description	item_price
563	56" Blue Freen	\$3.50
851	Spline End (Xtra Large)	\$0.25
652	3" Red Freen	\$12.00

FIGURA 1.15





## 3.2. Estandar del Framework

Al momento de diseñar una base de datos sujeta a las formas normales, es importante tener en cuenta los siguientes estándares, para que la instalación del Framework Semilla se ejecute correctamente.

- Los nombres de las tablas se recomienda sean en plural y se designarán de la siguiente manera:

**tbl\_nombre\_tabla\_plural**

Ejemplo: **tbl\_solicitudes**

- Las claves primarias de las tablas serán nombradas en singular seguido con el nombre de la tabla:

**id\_nombre\_tabla\_singular**

Ejemplo: **id\_solicitud**

- Se sugiere que todos los campos de las tablas sean nombrados en singular.
- No se puede repetir el nombre de un campo de una tabla en otra. Si existe por ejemplo una tabla **tbl\_trabajadores** que tiene un campo *nombre*, y existe otra tabla **tbl\_estados** con un campo *nombre*; se recomienda no repetir el mismo '**nombre**' en ambas tablas, para ello en la tabla **tbl\_trabajadores** se puede designar **nombre\_trabajador** y en la **tbl\_estados** por **estado**. Veamos:

id_trabajador
nombre_trabajador
.
.
.

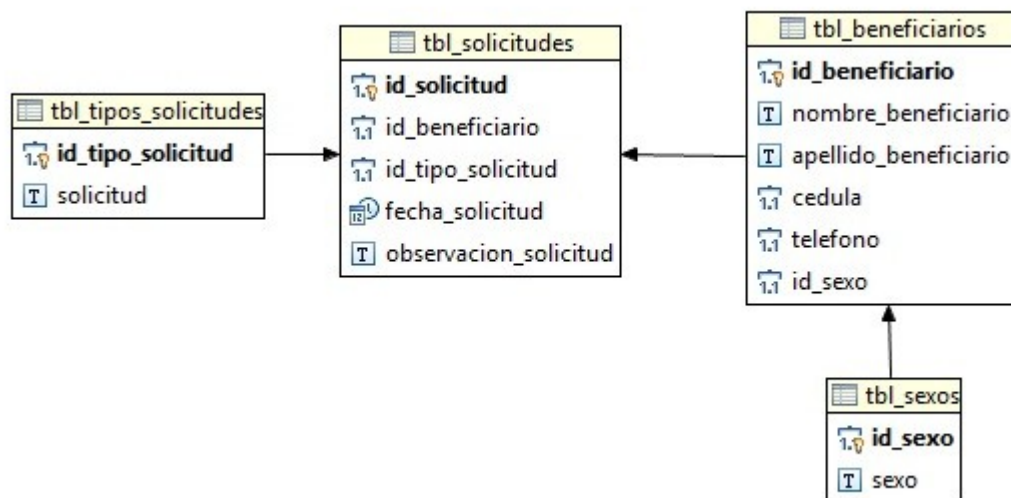
tbl\_trabajadores

id_estado
Estado
.
.
.

tbl\_estdos



### 3.3. Ejemplo





## ***4. MVC***



---

## ***4.1. Modelo-Vista-Controlador (MVC)***

- Este patrón fue descrito por primera vez por Trygve Reenskaug en 1979, y la implementación original fue realizada en Smalltalk en los laboratorios Xerox.
- Es un patrón de arquitectura de software que separa los datos de una aplicación (Modelo), la interfaz de usuario (Vista), y la lógica de control (Controlador) en tres componentes distintos.
- El patrón de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

## ***4.2. ¿Por qué utilizar MVC?***

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y el mantenimiento. MVC esta para ayudarnos a crear aplicaciones con mayor calidad.



---

## ***4.3. Modelo***

Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

## ***4.4. Vista***

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite **mostrar la salida**.



---

## **4.5. Controlador**

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como Insertar, Buscar, Consultar, Eliminar etc.

Es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

La "lógica de negocio": 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. Este es un concepto de desarrollo de software en general.



## 4.6. Ejemplos que Demuestran Ventajas en el MVC

- Aunque no tenga nada que ver, comencemos con algo tan sencillo como son el HTML y las CSS. Al principio, en el HTML se mezclaba tanto el contenido como la presentación. Es decir, en el propio HTML tenemos etiquetas como "font" que sirven para definir las características de una fuente, o atributos como "bgcolor" que definen el color de un fondo. El resultado es que tanto el contenido como la presentación estaban juntos y si algún día pretendíamos cambiar la forma con la que se mostraba una página, estábamos obligados a cambiar cada uno de los archivos HTML que componen una web, tocando todas y cada una de las etiquetas que hay en el documento. Con el tiempo se observó que eso no era práctico y se creó el lenguaje CSS, en el que se separó la responsabilidad de aplicar el formato de una web.
- Al escribir programas en lenguajes como PHP, cualquiera de nosotros comienza mezclando tanto el código PHP como el código HTML (e incluso el Javascript) en el mismo archivo. Esto produce lo que se denomina el **"Código Espagueti"**. Si algún día pretendemos cambiar el modo en cómo queremos que se muestre el contenido, estamos obligados a repasar todas y cada una de las páginas que tiene nuestro proyecto. Sería mucho más útil que el **HTML estuviera separado del PHP**.
- Si queremos que en un equipo intervengan perfiles distintos de profesionales y trabajen de manera autónoma, como diseñadores o programadores, ambos tienen que tocar los mismos archivos y el diseñador se tiene necesariamente que relacionar con mucho código en un lenguaje de programación que puede no serle familiar, siendo que a éste quizás solo le interesan los bloques donde hay HTML. De nuevo, sería mucho más fácil la **separación** del código.
- Durante la manipulación de datos en una aplicación es posible que estemos accediendo a los mismos datos en lugares distintos. Por ejemplo, podemos acceder a los datos de un artículo desde la página donde se muestra éste, la página donde se listan los artículos de un manual o la página de *backend* donde se administran los



---

artículos de un sitio web. Si un día cambiamos los datos de los artículos (alteramos la tabla para añadir nuevos campos o cambiar los existentes porque las necesidades de nuestros artículos varían), estamos obligados a cambiar, página a página, todos los lugares donde se consumían datos de los artículos. Además, si tenemos el código de acceso a datos disperso por decenas de lugares, es posible que estemos repitiendo las mismas sentencias de acceso a esos datos y por tanto no estamos **reutilizando código**.

Quizás te hayas visto en alguna de esas situaciones en el pasado. Son solo son simples ejemplos, habiendo decenas de casos similares en los que resultaría útil aplicar una arquitectura como el MVC, con la que nos obliguemos a separar nuestro código atendiendo a sus responsabilidades.

Ahora que ya podemos tener una idea de las ventajas que nos puede aportar el MVC, analicemos las diversas partes o conceptos en los que debemos separar el código de nuestras aplicaciones.





---

## 4.7. MVC en el Framework Semilla



**Controlador:** MVC



**Documentación:** contendrá toda la información del sistema, técnica y funcional.



**Inc:** contendrá los archivos de configuración del sistema.



**Instalar:** contiene el instalador del Framework Semilla.



**Js:** contendrá los archivos de JavaScript, librerías y plugin a ser utilizados.



**Log:** es un registro oficial de eventos de errores generados por la base de datos.



**Modelo:** MVC

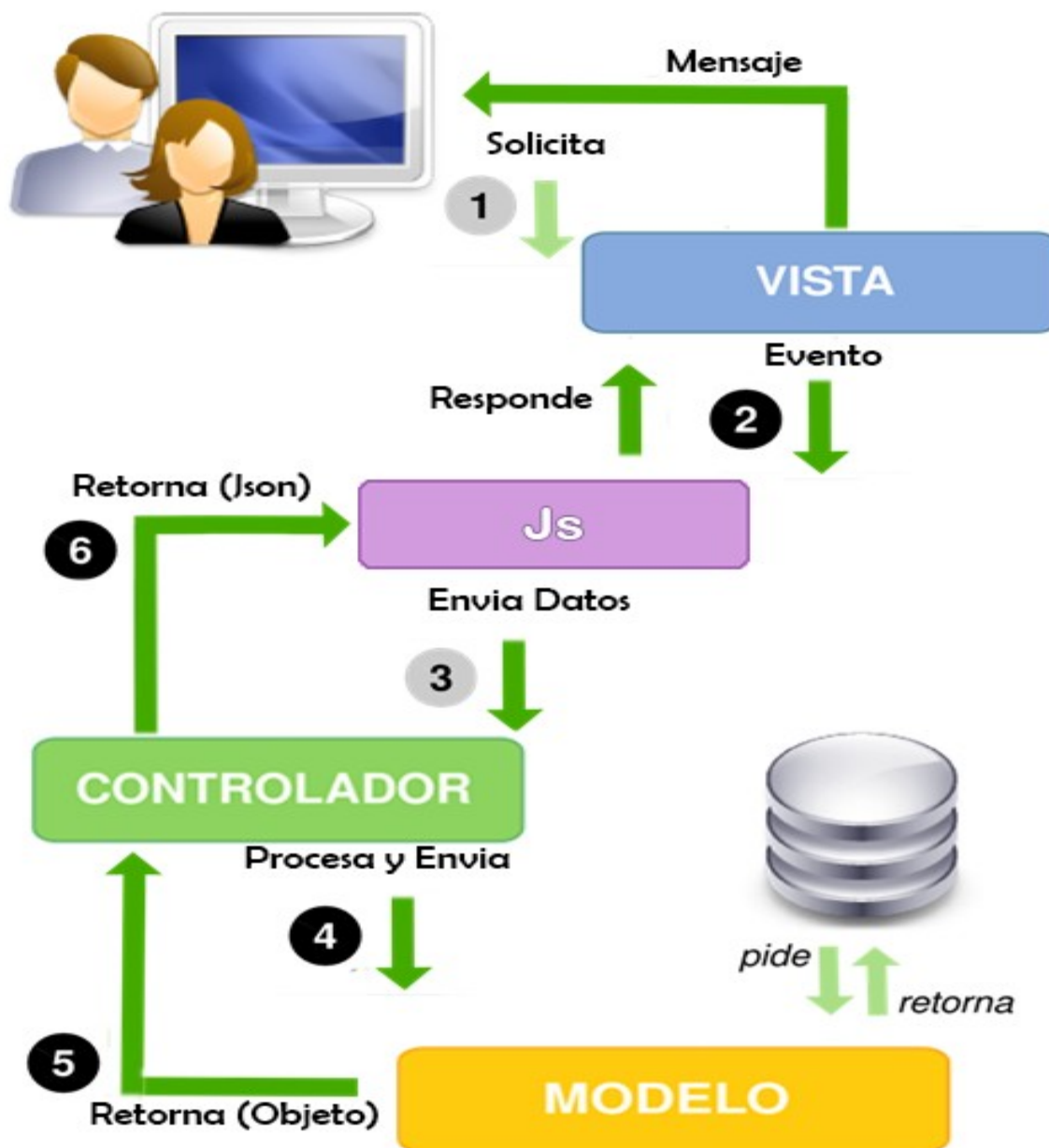


**Vistas:** MVC



**Prueba:** laboratorio de trabajo para realizar cualquier tipo de pruebas.

## 4.8. Diagrama MVC en el Framework Semilla





## ***5. POO***



---

## 5.1. ¿Qué es POO?

A partir de PHP 5, el modelo de objetos ha sido reescrito para permitir un mejor rendimiento y con más características, entre las características están la inclusión de la visibilidad, las clases abstractas y clases y métodos finales, métodos mágicos adicionales, interfaces, clonación y tipos sugeridos.

PHP trata los objetos de la misma manera como referencias o manejadores, lo que significa que cada variable contiene una referencia de objeto en lugar de una copia de todo el objeto.

POO es cuando planteamos clases y definimos objetos de las mismas (Este es el objetivo del tutorial, aprender la metodología de programación orientada a objetos y la sintaxis particular de PHP 5 para la POO).

### **Conceptos Básicos de Objetos:**

Un objeto es una entidad independiente con sus propios datos y programación. Las ventanas, menús, carpetas de archivos pueden ser identificados como objetos; el motor de un auto también es considerado un objeto, en este caso, sus datos (atributos) describen sus características físicas y su programación (métodos) describen el funcionamiento interno y su interrelación con otras partes del automóvil (también objetos).

El concepto renovador de la tecnología Orientación a Objetos es la suma de funciones a elementos de datos, a esta unión se le llama encapsulamiento. Por ejemplo, un objeto página contiene las dimensiones físicas de la página (ancho, alto), el color, el estilo del



borde, etc, llamados atributos. Encapsulados con estos datos se encuentran los métodos para modificar el tamaño de la página, cambiar el color, mostrar texto, etc. La responsabilidad de un objeto página consiste en realizar las acciones apropiadas y mantener actualizados sus datos internos. Cuando otra parte del programa (otros objetos) necesitan que la página realice alguna de estas tareas (por ejemplo, cambiar de color) le envía un mensaje. A estos objetos que envían mensajes no les interesa la manera en que el objeto página lleva a cabo sus tareas ni las estructuras de datos que maneja, por ello, están ocultos. Entonces, un objeto contiene información pública, lo que necesitan los otros objetos para interactuar con él e información privada, interna, lo que necesita el objeto para operar y que es irrelevante para los otros objetos de la aplicación.

## 5.2. POO en el Framework Semilla

En el Framework Semilla las clases son creadas de manera automática con una técnica de programación denominada ORM (Object-Relational mapping), mapeo objeto-relacional, que nos permite tener una capa de abstracción de datos con ADOdb de php, implementando métodos mágicos de php, en su construcción.

Veamos un ejemplo de una clase generada:

### SINTAXIS – CLASE PHP

```
<?php session_start( );  
  
class tbl_fases extends general {  
    public $bd;  
    public $campos="**";  
    public $where;  
    private $ObjetSQL;  
    private $tabla;  
    private $propiedades = array();
```



```
public function __construct( $id_fase=NULL,$fase=NULL) {  
    $main = main::getInstance();  
    $MotorSQL = $main->motor."_sql";  
    include_once(MODELO."abstraccion".DS."db".DS."{$main->motor}-sql.lang.php");  
    $this->ObjetSQL = new $MotorSQL();  
    $this->bd = db::getInstance();  
    $this->propiedades["id_fase"]=$id_fase;  
    $this->propiedades["fase"]=$fase;  
    $this->where=" id_fase = ".$this->bd->qstr($this->propiedades["id_fase"]);  
    $this->tabla="schema_sigid.tbl_fases";  
}  
  
public function __set( $nombre, $valor) { $this->propiedades[$nombre] = $valor; }  
  
public function __get( $nombre) { return $this->propiedades[$nombre]; }  
  
public function __call( $metodo, $args) {  
    if ($args) { $method = array(&$this->bd, $metodo);  
    $a = call_user_func_array($method, $args);  
    }else { $a = $this->bd->$metodo(); }  
    return $a;  
}  
  
public function insertar () {  
    try { $retorno=$this->id_fase=$this->GenID('schema_sigid.tbl_fases_id_fase_seq');  
    $record=$this->AutoExecute($this->tabla,$this->propiedades,'INSERT');  
    }  
    catch (exception $e) { $retorno=$this->ErrorMsg(); }  
    return $retorno;  
}
```



```
public function actualizar (){
    try { $record=$this->AutoExecute($this->tabla,$this->propiedades,'UPDATE',$this->where);
    if ($record){ $retorno=$this->id_fase; }
    else{ $retorno=false; }
    }

    catch (exception $e) { $retorno=$this->ErrorMsg(); }

return $retorno;
}

public function eliminar (){ $ObjetSQL=&$this->ObjetSQL;

    $sql.=sprintf($ObjetSQL->delete,$this->tabla);
    $sql.=sprintf($ObjetSQL->where,$this->where);

    try {
        $record=$this->Execute($sql);
        if ($record){ $retorno=1; }
        else{ $retorno=false; }
    }

    catch (exception $e) { $retorno=$this->ErrorMsg(); }

return $retorno;
}

public function listar ( $campos=false){ $ObjetSQL=&$this->ObjetSQL;

    $this->tabla_!= $this->tabla;

    $sql.=sprintf($ObjetSQL->select,$this->campos,$this->tabla_!);

    if ($campos){ $sql.=sprintf($ObjetSQL->where,$this->where); }

    $sql.=$this->v_limit;
    //echo $sql; exit;
    $record = $this->Execute($sql);

    if ($record){ $retorno=$record->GetRows(); }
    else{ $retorno=false; }
```



```
return $retorno;
}

public function total_registros () { $ObjetSQL=&$this->ObjetSQL;
                                     $this->campos_t=" count(*) as total_registros ";
                                     $sql=sprintf($ObjetSQL->select,$this->campos_t,$this->tabla);

                                     if($this->where){ $sql.=sprintf($ObjetSQL->where,$this->where); }

                                     try { $record = $this->Execute($sql);
                                     if ($record){ $retorno=$record->GetRows();
                                     $retorno=$retorno[0]["total_registros"]; }
                                     else{ $retorno=false; }
                                     }

                                     catch (exception $e) { $retorno=$this->ErrorMsg(); }

return $retorno;
}

public function limit ($inicio,$fin){ $ObjetSQL=&$this->ObjetSQL;
if ($_SESSION["where"]!="") {$sql=true;}

                                     else {$sql=false;}

$this->v_limit=sprintf($ObjetSQL->limit,$inicio,$fin);
//echo $sql; exit;

$retorno = $this->listar($sql);
return $retorno;
}

public function buscar ($campos=true){
    if (($campos) && ($_SESSION["where"]=="")) {
        $no_buscar = array("id_usuario","usuario","computadora","fecha_hora_sis");
        foreach ($this->propiedades as $key => $valor)
```





```
{
    if(($valor) && !(in_array($key, $no_buscar))) {
        if (substr_count($key,"id_")) { $where.="$key in ($valor) AND "; }
        else { $where.="$key LIKE'%$valor%' AND "; }
    }
}

$where.="false";
$where=str_replace(" AND false","", $where);
$this->where=$where;
$_SESSION["where"]=$this->where;
}

else
    { $this->where=$_SESSION["where"]; }

//echo $this->where; exit;
$retorno = $this->listar(true);
return $retorno;
}
```

```
public function ver_opciones ($ver=NULL){ $campos=$this->listar();
```

```
    if ($campos) {
        foreach($campos as $index => $valor)
        {
            if ($valor["id_fase"]==$ver){ $option.="<option value='".$valor["id_fase"]."' selected>".$valor["fase"]."</option>";}
            else { $option.="<option value='".$valor["id_fase"]."' >".$valor["fase"]."</option>"; }
        }
    }
    else {$option = false;}

return $option;
}
```

```
public function ver_checkbox ($buscado) { $campos=$this->listar();
```



```
if ($campos) {  
    foreach( $campos as $index => $valor)  
    {  
        if (in_array($valor["id_fase"], $buscado))  
        { $option.="<input type='checkbox' value='".$valor["id_fase"]." checked='checked' />".$valor["fase"]."<br />"; }  
        else { $option.="<input type='checkbox' value='".$valor["id_fase"]." />".$valor["fase"]."<br />"; }  
    }  
}  
else { $option = false; }  
  
    return $option;  
}  
  
public function valor_campo ($campo) { $campos=$this->listar(true);  
    return $campos[0][$campo];  
}  
}  
?>
```

Al utilizar el framework semilla con NetBeans permite hacer uso de la ayuda cuando haces uso de algún método o atributo de las clases generadas.

Las tablas mapeadas por el ORM y generadas por el framework semilla todas las crea de la misma forma, ejemplo:

DEFINICIÓN	SINTAXIS
El nombre de la clase	<i>class tbl_periodos extends general</i>
Todos los atributos que maneja la clase	<i>public \$bd; public \$campos="*"; public \$where; private \$ObjetSQL;</i>



	<pre>private \$tabla; private \$propiedades = array();</pre>
Todos los métodos de la clase	<pre>__construct(\$id_periodo=NULL,\$nombre_periodo=NULL, \$fecha_inicio_periodo=NULL,\$fecha_fin_periodo=NULL, \$estatu_periodo=NULL) __set(\$nombre, \$valor) __get(\$nombre) __call(\$metodo, \$args) insertar() actualizar() eliminar() listar(\$campos=false) total_registros() limit(\$inicio,\$fin) buscar(\$campos=true) ver_opciones(\$ver=NULL) ver_checkbox(\$buscado) valor_campo(\$campo)</pre>



## ***6. INSTALACIÓN***



El framework semilla posee un instalador con una interfaz gráfica amigable que guiará al usuario a través de todo el proceso, la instalación es una aplicación web la cual puede ser usada en cualquier sistema operativo. A continuación se detalla las aplicaciones con las que debe contar el computador y/o portátil.

## 6.1. ¿Qué se Debe Tener?

### *Aplicaciones Necesarias (Windows)*

- **Servidor HTTP:** será el que nos permita ejecutar aplicaciones web desde cualquier explorador, funcionando en la PC como un servidor virtual. Para Windows existe un paquete completo que nos proporciona (Apache, MySQL y PHP) llamado wampserver, también existe XAMP que es un servidor independiente de plataforma, software libre, (Apache, MySQL , PHP Perl).

WANSERVER: <http://www.wampserver.com/en/>

XAMPP: <https://www.apachefriends.org/es/>





- **Motor de Base de datos:** el framework semilla puede trabajar con dos motores de base de datos, los cuales son Postgresql y Mysql. En el caso de Mysql ya viene instalado por defecto con la paquetería de los Servidores HTTP antes mencionados y proporciona una interfaz web para el manejo de la misma phpMyadmin. Para trabajar con postgresql se debe tener instalado el motor de base de datos postgres y una manejador de base de datos postgres, el mas común es Pgadmin.

PgAdmin: <http://www.pgadmin.org/>

PhpPgAdmin: <http://phppgadmin.sourceforge.net/doku.php?id=start>



- **Explorador:** Un navegador o navegador web, o browser, es un software que permite el acceso a Internet, interpretando la información de distintos tipos de archivos y sitios web para que éstos puedan ser visualizados. Ya que el framework semilla es una aplicación se debe contar con un explorador se recomienda Mozilla Firefox en su ultima versión.



- **Extra: Plugin Firefox:** es una herramienta muy popular para el desarrollo web la cual



permite: Inspeccionar HTML y modificar el estilo y el diseño en tiempo real, Identificar un elemento en una página web con facilidad y precisión y muchas otras útiles al momento del programar.

<https://addons.mozilla.org/es/firefox/addon/firebug/>





---

## Aplicaciones Necesarias (Linux):

- **Servidor HTTP:** sera el que nos permita ejecutar aplicaciones web desde cualquier explorador, funcionando en la PC como un servidor virtual. El mas común en software libre es Apache2, el cual se puede instalar desde la consola con privilegios de administrador.

Instalación de apache2:

```
# apt-get install apache2
```



- **Motor de Base de datos:** el framework semilla puede trabajar con dos motores de base de datos, los cuales son Postgresql y Mysql. Se deben instalar los mismos y luego un manejador de base de datos para Postgresql y uno para Mysql el cual proporcionara una interfaz gráfica para el manejo de la base de datos.

- **Postgresql:**

**PgAdmin:** <http://www.pgadmin.org/>

**PhpPgAdmin:** <http://phppgadmin.sourceforge.net/doku.php?id=start>







#### ■ **Mysql:**

<http://dev.mysql.com/downloads/mysql/>



## **6.2. Primeros Pasos**

Una vez que la PC posee todas las aplicaciones necesarias y con la base de datos generada en cualquiera de los motores antes mencionados, se procede a instalar el framework:

- 1.-** Se procede a copiar la carpeta que contiene el framework en nuestro directorio de servidor web (www).
- 2.-** Extraemos el archivo que se encuentra comprimido en .zip



framework\_semilla\_3\_5.zip  
15,3 MiB

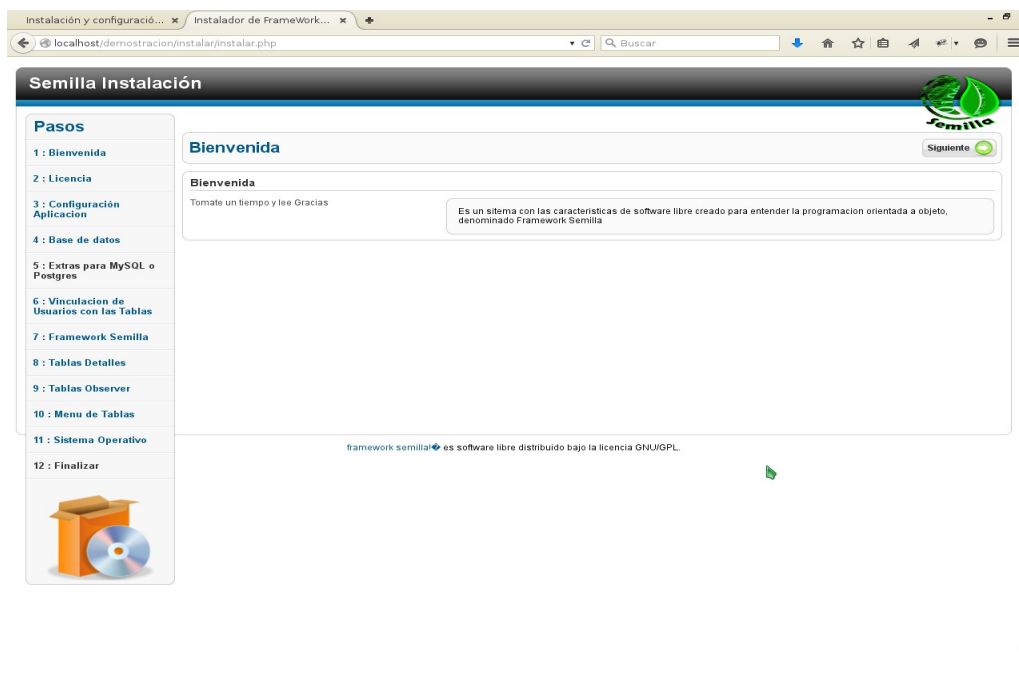
- 3.-** Cambiamos el nombre de la carpeta extraída por el nombre o siglas de nuestro sistema.



demostracion

#### 4.- Accedemos desde un explorador con la ruta **http://localhost/demostracion/instalar**

Una vez que se accede a la ruta se debe visualizar una ventana de bienvenida al instalador de Framework semilla. Si estamos trabajando en el entorno de Linux se debe dar privilegios a la carpeta del framework una vez que sea renombrada.



#### 5.- Una vez presionado siguiente se muestra unos términos y condiciones (Licencia Pública



## General GNU)



6.- Esta pantalla es para configurar nuestro sistema. El servidor por defecto es localhost, el nombre de la carpeta raíz no se puede modificar por defecto el instalador lo genera, en el campo nombre de sistema sera el nombre para mostrar en el sistema y el combo de Layout son todas las plantillas que posee el framework semilla para las vista de los usuarios.



**Semilla Instalación**

**Pasos**

- 1 : Bienvenida
- 2 : Licencia
- 3 : Configuración Aplicación
- 4 : Base de datos
- 5 : Extras para MySQL o Postgres
- 6 : Vinculación de Usuarios con las Tablas
- 7 : Framework Semilla
- 8 : Tablas Detalles
- 9 : Tablas Observer
- 10 : Menu de Tablas
- 11 : Sistema Operativo
- 12 : Finalizar

**Configuración Aplicación**

Configuración Aplicación

Configuración de la Aplicación

Servidor  
 recuerde colocar http://nombre\_ser\_servidor

Carpeta Raiz del sistema  
 recuerde no colocar espacios

nombre del sistema  
 este nombre sera utilizado como titulo de la aplicacion

Layout por defecto para toda la aplicacion  
 es recomendable dejar esta seccion con el valor de defecto si no cuenta con otro layout

framework semilla es software libre distribuido bajo la licencia GNU/GPL.

7.- En este paso se configura la conexión a la Base de Datos, el tipo de base de datos puede ser PostgreSQL o MySQL, el nombre del Host (es el Host donde se encuentre alojada la base de datos), el usuario que tiene acceso a esa base de datos y contraseña para conectarse a la base de datos.

**Semilla Instalación**

**Pasos**

- 1 : Bienvenida
- 2 : Licencia
- 3 : Configuración Aplicación
- 4 : Base de datos
- 5 : Extras para MySQL o Postgres
- 6 : Vinculación de Usuarios con las Tablas
- 7 : Framework Semilla
- 8 : Tablas Detalles
- 9 : Tablas Observer
- 10 : Menu de Tablas
- 11 : Sistema Operativo
- 12 : Finalizar

**Configuración de Base de Datos**

Configuración de Base de Datos

Configuración de Base de Datos

Tipo Base de Datos  
 seleccione el motor de base de datos

Nombre del Host  
 Esto es por lo general "localhost"

Usuario  
 O algo como "root" o un nombre de usuario dado por el host

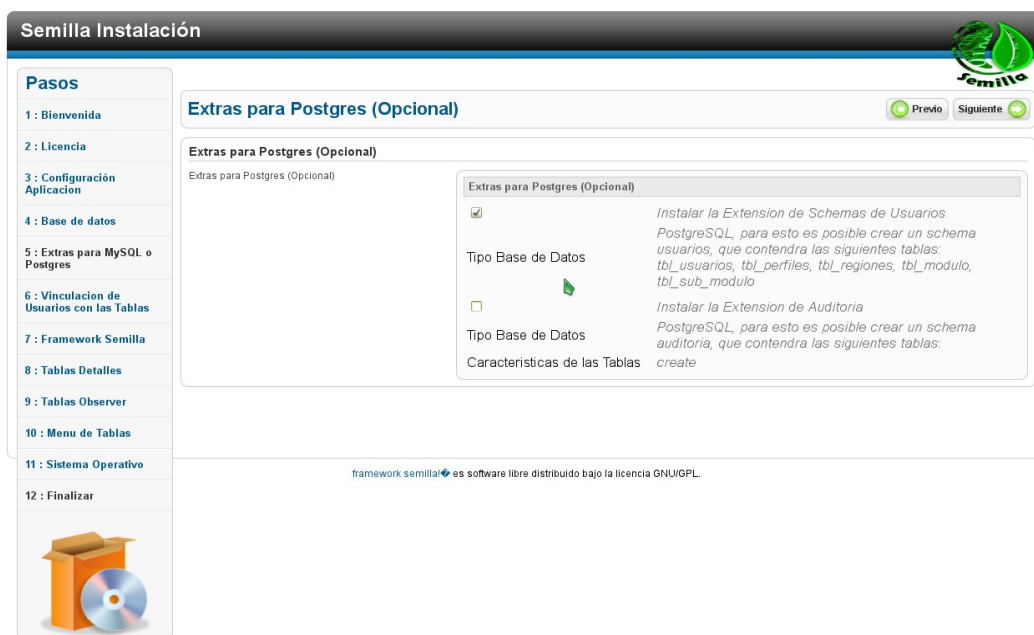
Contraseña  
 Es para conectarte a la base de datos

Nombre de la base de datos  
 el nombre de la base de datos

framework semilla es software libre distribuido bajo la licencia GNU/GPL.




**8.-** En caso de trabajar con una base de datos en Postgresql el instalador de framework proporciona configuraciones extra para este motor de Base de datos. Este extra permite Instalar extensión para usuarios: esta crea en la base de datos un nuevo schema llamado schema\_usuarios, en el cual se generaran unas series de tablas para controlar los usuarios que acceden al sistema con sus privilegios, controlar módulos y sub-módulos del sistema a crear.





9.- El paso numero 6 de las instalación nos permite vincular por medio de la base de datos el usuario a cualquiera de las tablas, este es útil ya que se guardada en los registros de las tablas seleccionadas el usuario que esta realizando un cambio a dicho registros, de modo que pueda servir para saber quien modifico un registro mediante el sistema.

Semilla Instalación



Siguiente

Pasos

1 : Bienvenida

2 : Licencia

3 : Configuración Aplicación

4 : Base de datos

5 : Extras para MySQL o Postgres

6 : Vinculación de Usuarios con las Tablas

7 : Framework Semilla


8 : Tablas Detalles

9 : Tablas Observer

10 : Menu de Tablas

11 : Sistema Operativo

12 : Finalizar



Bienvenida

Bienvenida

Tomate un tiempo y lee gracias

Nombre	Vincular
public.custom_fields_lists	<input type="checkbox"/>
schema_sigid.tbl_evaluaciones_rangos	<input type="checkbox"/>
schema_sigid.tbl_analistas_ubicaciones	<input type="checkbox"/>
schema_sigid.tbl_cursos	<input type="checkbox"/>
schema_sigid.tbl_estatus_personales	<input type="checkbox"/>
schema_sigid.tbl_reposos	<input type="checkbox"/>
schema_sigid.tbl_analistas	<input type="checkbox"/>
schema_sigid.tbl_regiones	<input type="checkbox"/>
public.tbl_trabajadores	<input type="checkbox"/>
schema_sigid.tbl_fechas_fases	<input type="checkbox"/>
schema_sigid.tbl_ubicaciones_administrativas	<input type="checkbox"/>
schema_sigid.tbl_grupos	<input type="checkbox"/>
schema_sigid.tbl_supervisor	<input type="checkbox"/>
schema_sigid.tbl_odis	<input type="checkbox"/>
schema_sigid.tbl_trabajadores	<input type="checkbox"/>
schema_sigid.tbl_vacaciones	<input type="checkbox"/>
schema_sigid.tbl_tipos_evaluaciones	<input type="checkbox"/>
schema_sigid.tbl_tipo_personal	<input type="checkbox"/>
public.custom_fields_values	<input type="checkbox"/>
schema_sigid.tbl_personales_supervisado	<input type="checkbox"/>
schema_sigid.tbl_fases	<input type="checkbox"/>
schema_sigid.tbl_adiestramiento	<input type="checkbox"/>



**10.-** En el paso numero 7 del instalador se muestran todas las tablas de la base de datos seleccionada, este paso es para generar el MVC del sistema, se podrá activar o desactivar las tablas que se desee y las características del MVC que se requieran.

**Semilla Instalación**

**Pasos**

- 1: Bienvenida
- 2: Licencia
- 3: Configuración Aplicación
- 4: Base de datos
- 5: Extras para MySQL o Postgres
- 6: Vinculación de Usuarios con las Tablas
- 7: Framework Semilla
- 8: Tablas Detalles
- 9: Tablas Observer
- 10: Menu de Tablas
- 11: Sistema Operativo
- 12: Finalizar

**Configuración Aplicación**

**Bienvenida**

Selecciona las Tablas a Programar

Nombre	<input checked="" type="checkbox"/> Modelo (Class)	<input checked="" type="checkbox"/> Vista (Formulario)	<input checked="" type="checkbox"/> Controlador (Logica negocio)	<input checked="" type="checkbox"/> Listar (Ver datos)
<input checked="" type="checkbox"/> public custom_fields_lists	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_evaluaciones_rangos	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_analistas_ubicaciones	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_cursos	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_estatus_personales	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_reposos	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_usuarios.tbl_estatus_usuarios	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_analistas	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_regiones	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> public.tbl_trabajadores	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_fechas_fases	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_usuarios.tbl_modulos_perfiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_ubicaciones_administrativas	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_grupos	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_supervisor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_odis	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_trabajadores	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_vacaciones	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>



**11.-** El paso numero 11 de la instalación de Framework Tabla Detalles. Esta referido a aquellas tablas que dependan de otras de manera inmediata, la seleccionada como tabla padre sera la primera en mostrar el formulario, una vez que se realice la acción sobre esta tabla (Insertar), el sistema automáticamente lo direccionara al formulario de la segunda tabla o tabla hija, permitiendo realizar inserciones de registros en esta segunda.

**Semilla Instalación**

**Pasos**

- 1 : Bienvenida
- 2 : Licencia
- 3 : Configuración Aplicación
- 4 : Base de datos
- 5 : Extras para MySQL o Postgres
- 6 : Vinculación de Usuarios con las Tablas
- 7 : Framework Semilla
- 8 : Tablas Detalles
- 9 : Tablas Observer
- 10 : Menu de Tablas
- 11 : Sistema Operativo
- 12 : Finalizar

**Bienvenida**

Tomate un tiempo y lee gracias

Tabla padre: custom\_fields\_lists

Tabla hija (Detalle): custom\_fields\_lists

Procesar

framework semilla es software libre distribuido bajo la licencia GNU/GPL





**12.-** El paso numero 12 Tabla Observar. El patrón Observer que nos proporciona una vista en tiempo de real de las acciones que se ejercen sobre una tabla de la base de datos, mostrando en el formulario un listado del contenido que posee la tabla en el momento actual. Se selecciona la tabla en el listado desplegable y se presiona el botón procesar.

**Semilla Instalación**

**Pasos**

- 1: Bienvenida
- 2: Licencia
- 3: Configuración Aplicación
- 4: Base de datos
- 5: Extras para MySQL o Postgres
- 6: Vinculación de Usuarios con las Tablas
- 7: Framework Semilla
- 8: Tablas Detalles
- 9: Tablas Observer
- 10: Menu de Tablas
- 11: Sistema Operativo
- 12: Finalizar

**Bienvenida**

Tómese un tiempo y lee gracias

Tabla Observer:

framework semilla es software libre distribuido bajo la licencia GNU/GPL

### Ejemplo de Observer:

#### Sin Observer

**INSERTAR - Fases**

Fase:

#### Con Observer

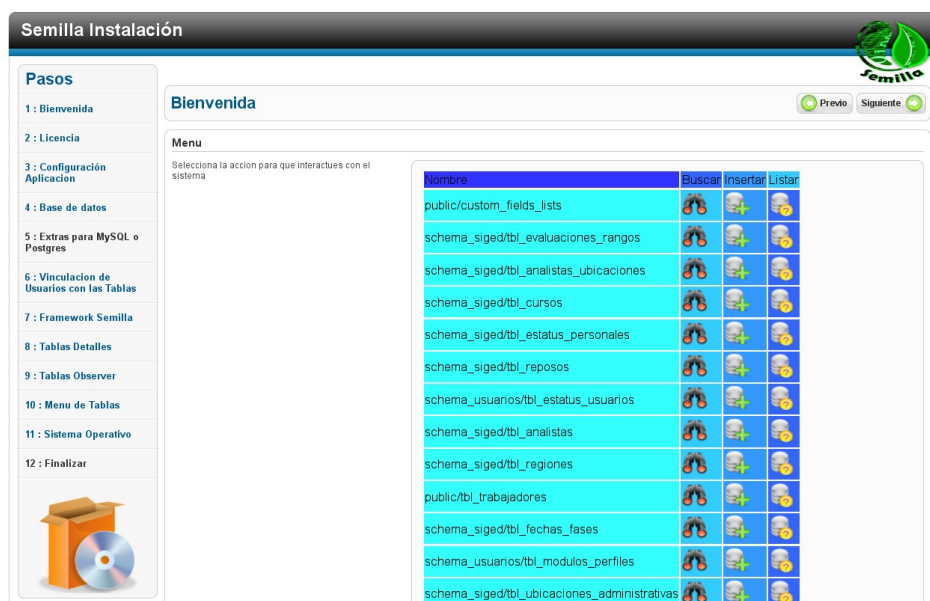
**INSERTAR - Fases**

Fase:

FASE	FASE	ACCION
1	Asignación y Registro de ODIS	<input type="button" value="Actualizar"/> <input type="button" value="Eliminar"/>
2	Revisión y Validación de Personal y ODIS	<input type="button" value="Actualizar"/> <input type="button" value="Eliminar"/>
3	Evaluación de Personal	<input type="button" value="Actualizar"/> <input type="button" value="Eliminar"/>



**13.-** El paso numero 13 del instalado muestra todas las tablas que se le creo el MVC, con tres opciones por cada una de ellas: Buscar, Insertar y Listar. Al presionar cada uno de estos iconos se podra ver el formulario con la acción solicitada en el caso de buscar o Insertar y un Listado con los registros que existen la tabla en el caso de Listar.





## 6.3. Generar MVC de Nuevas Tablas

Una vez que tengamos instalado el Framework y necesitemos realizar cambios a la base de datos se procederá de la siguiente forma.

**1.-** Modificar la Base de datos. Se pueden crear nuevas tablas o modificar campos existentes. En ambos casos se debe generar nuevamente el MVC de la tabla afectada.

**2.-** Ingresar en la ruta de instalación del Framework

<http://localhost/demostracion/instalar>

**3.-** Seleccionar el paso Numero 7 en el Menú de instalación.

**7 : Framework Semilla**

**4.-** Deseleccionar todas las tablas excepto las que se hallan modificado.

Nombre	<input checked="" type="checkbox"/> Modelo (Class)	<input checked="" type="checkbox"/> Vista (Formulario)	<input checked="" type="checkbox"/> Controlador (Logica negocio)	<input checked="" type="checkbox"/> Listar (Ver datos)
<input type="checkbox"/> public.custom_fields_lists	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> schema_sigid.tbl_evaluaciones_rangos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> schema_sigid.tbl_analistas_ubicaciones	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> schema_sigid.tbl_cursos	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> schema_sigid.tbl_estatus_personales	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> schema_sigid.tbl_reposos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> schema_usuarios.tbl_estatus_usuarios	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> schema_sigid.tbl_analistas	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Este proceso generara nuevamente los archivos Vista, Js, Controlador y Modelo para la tabla selecciona los archivos que ya existían se conservaran con la fecha actual por seguridad.



## ***7. LIBRERÍAS DEL FRAMEWORK***



En el framework vienen instaladas las siguientes librerías por defecto:

- FPDF: es una librería para la generación de reportes en formato pdf, para mayor información ***<http://www.fpdf.org/es/doc/index.php>***.
- PHPMAILER: es una librería para el envío de correos electrónicos, para mayor información ***[http://www.lubrimor.com/admin/imagenes/producto/03\\_04\\_2012\\_09\\_10\\_06\\_811219648.pdf](http://www.lubrimor.com/admin/imagenes/producto/03_04_2012_09_10_06_811219648.pdf)***.
- JQUERY.VALIDATE: controla las validaciones de los formularios, para mayor información, consulte: ***<http://jqueryvalidation.org/valid/>***



## ***8. PREGUNTAS FRECUENTES***



---

## ■ ¿Como puedo cambiar la Interfaz?

Resp.: Para cambiar la interfaz se debe acceder al archivo: inc/config.sistema.php, el cual es un archivo que contiene las variables de configuración del sistema. Se debe sustituir la variable \$default\_layouts con el nombre de uno de los layouts que posee el Framework Semilla.

```
$default_layouts="menpet";  
$default_layouts="bootstrap";  
$default_layouts="default";  
$default_layouts="menu_superior";
```

## ■ ¿Como incorporo un nuevo campo a una tabla después de instalado el Framework?

Resp.: si se requiere colocar un nuevo campo en una tabla de la base de datos después que se ha instalado el framework, se presentan dos posibilidades, de forma Automática o de forma Manual.

Automática: entrar en la ruta de instalación para generar el Nuevo MVC de la tabla afectada, cabe destacar que los avances que se hayan realizado no se perderán pues al generar nuevos archivos el Framework respalda los antiguos con el nombre de "nombre\_archivo old fecha". El procedimiento detallado se encuentra en el Capítulo de Instalación -Generar MVC de Nuevas Tablas

Manual: agregar un campo de forma manual, se debe agregar el campo en el siguiente orden.

1. Agregar el campo en la base de datos.
2. Agregar el campo en la clase correspondiente a la tabla afectada.



Ruta: modelo/schema\_nombre/class\_tbl\_nombre\_tabla.php

La función que se debe afectar es `public function __construct($nuevo_campo=NULL)`

Agregar la Propiedad. `$this->propiedades["nuevo_campo"]=$nuevo_campo;`

3. Agregar el campo en el controlador de la tabla afectada.

Ruta: controlador/schema\_nombre/tbl\_nombre\_tabla.php

Agregar el campo en el Objeto perteneciente a la clase y tabla afectada

`$_REQUEST["nuevo_campo"]`

```
require_once("../modelo/schema_sigid/class_tbl_cursos.php"); # clase del modelo
$obj_tbl_cursos = new tbl_cursos($_REQUEST["id_curso"],$_REQUEST["id_adiestramiento"]);
```

4. Agregar el campo en el archivo js de la tabla en caso de ser necesario (en la validaciones o Rules para campo requeridos o en validaciones que se requiera.)

5. Agregar el campo en la vista, de ser necesario.

#### ■ ¿Como fusionar dos vista en un solo formulario?

Resp.: Para realizar esta combinación se debe copiar los datos de un formulario y pegarlos en el formulario que se seleccione como primario. Además se debe copiar los enlaces de las clases que se utilizan del archivo de origen al archivo primario.

```
require_once("../modelo/schema_sigid/class_tbl_cursos.php"); # clase del modelo
$obj_tbl_cursos = new tbl_cursos($_REQUEST["id_curso"],$_REQUEST["id_adiestramiento"]);
```

#### ■ ¿Como insertar en 2 tablas desde un controlador?

Resp.: Suele usarse en el caso que se fusionen dos formularios. Para realizar dicha inserción se debe seleccionar el controlador principal o archivo controlador que corresponda a la tabla





principal de trabajo.

1.- Se incluye en el controlador la clase y se declara el objeto de la tabla o tablas que se desea manipular.

```
require_once("../modelo/schema_siged/class_tbl_cursos.php"); # clase del modelo  
$obj_tbl_cursos = new tbl_cursos($_REQUEST["id_curso"],$_REQUEST["id_adiestramiento"]);
```

2.- En el Case:"insertar" del controlador principal se puede manipular el objeto de la tabla secundaria o tablas que se necesite insertar.

3.- Realizar el llamado a la función insertar de dicha tabla secundaria

```
$_REQUEST["id_curso_evaluacion"]=$obj_tbl_cursos_evaluaciones->insertar();
```

#### ■ ¿Puedo construir mi propio Layout?

Resp.: Si se puede construir, se debe crear una carpeta con el nombre del layout en vista/layouts/nombre\_layout. Para construirlo se debe crear el contenido dentro de esta carpeta manteniendo el estándar que se aprecia en los otros layouts (footer, header) y crear las carpetas con el contenido css e imágenes que mostrar la nueva vista.

#### ■ ¿Puedo crear mi propio Controlador?

Resp.: Si solo se sugiere respetar la arquitectura de patrón MVC y crear el archivo en la carpeta controlador, sin olvidar incluir en este nuevo controlador las clases que se necesiten para su correcto funcionamiento.



## ***9. TIPS PARA EL MANEJO DEL FRAMEWORK SEMILLA***



- Verificar los permisos de los archivos cuando se realicen cambios (Linux)
- Al utilizar archivos RTF (Formato de texto enriquecido), evaluar las cabeceras o espacios en blanco, ya que puede hacer que el archivo se distorsione al abrirlo.
- Cuando las tablas se encuentran en schemas diferentes verificar las funciones listar que se encuentran en los archivos controladores. Esta función está conformada por Join, verificar las rutas de join.
- En los formularios existen select (Muestran los datos de una tabla), se pueden filtrar con un parámetro dentro del Where.
- No utilizar las clases, solo en caso que sea necesario con una función propia de la clases.
- Si se desea colocar valores por defecto, hacerlo desde el controlador.
- No utilizar acentos ni espacios en base de datos.
- Utilizar un entorno de desarrollo para la edición de código (Netbeans).
- Si utiliza campos de tipo:date en la base de datos, deberá activar los datepicker en los archivos .js correspondiente a dicha tabla que contiene el campo.
- Al incluir plugins verificar la compatibilidad con la librería js que incluye el Framework semilla.
- Se pueden crear funciones estándar para JQuery para ser utilizadas en cualquier archivo js. Las mismas deben estar alojadas en el archivo:  
*js/validaciones/configuracion.js*
- Se pueden crear funciones estándar para PHP para ser utilizadas en cualquier archivo controlador o en formularios dentro de sentencia PHP. Las mismas deben estar



alojadas en el archivo: *modelo/class\_general.php*.