Contenido

1.- Introducción.

1.1 Páginas web Dinámicas.

Se conoce con el nombre de página web dinámica a aquélla, cuyo contenido se genera a partir de lo que un usuario introduce en un web o formulario. El contenido de la página no está incluido en un archivo HTML como en el caso de las páginas web estáticas.

Contrariamente a las páginas estáticas, en las que su contenido se encuentra predeterminado, en las dinámicas la información aparece inmediatamente después de una solicitud hecha por el usuario. Una página dinámica permite visualizar la información contenida en una base de datos, así como almacenar y hacer actualizaciones de cierta

información a través de un formulario. Además se pueden manejar foros y el usuario tiene la posibilidad de cambiar a su gusto el diseño y el contenido de la página, entre otras cosas.

Para la creación de este tipo de páginas, además de etiquetas HTML es necesaria la utilización de algún lenguaje de programación que se ejecute del lado del servidor, así como la existencia de una base de datos.

Los lenguajes más utilizados para la generación de este tipo de páginas son:

- Perl CGI
- PHP
- JSP
- ASP

Los manejadores de bases de datos más comunes que pueden trabajar con páginas dinámicas son:

- PostgresSQL
- MySQL
- Oracle
- Microsoft SQL Server

Las páginas Web dinámicas ofrecen muchas ventajas a diferencia de las páginas Web estáticas, como una mayor interactividad con el usuario, mientras que al administrador le permiten una reducción en tiempo y costos, así como una mayor facilidad en el mantenimiento de un sitio.

Las aplicaciones más conocidas de las páginas web dinámicas son:

- Mostrar el contenido de una base de datos, con base en la información que solicita un usuario a través de un formulario de web.
- Actualizar el contenido de una base de datos.
- Generar páginas web de contenido estático.
- Mejorar la interacción entre el usuario y el sitio web.

1.2 Introducción a PHP.

Aunque existe una gran cantidad de lenguajes de entornos de desarrollo concebidos para internet, PHP se ha convertido en uno de los lenguajes, del lado del servidor, más ampliamente utilizados para el desarrollo de páginas dinámicas junto con lenguajes como ASP, JSP y Perl.

1.2.1 ¿Qué es PHP?

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Definiéndolo de una forma muy concisa PHP es un lenguaje interpretado de alto nivel embebido en HTML y ejecutado en el servidor.

Ejemplo introductorio:

```
<html>
      <head>
            <title>Ejemplo PHP</title>
      </head>
      <body>
            <?php echo "Hola, este es un ejemplo con PHP!"; ?>
      </body>
</html>
```

Como podemos ver en el ejemplo en vez de escribir un programa con muchos comandos para crear una salida en HTML, escribimos el código HTML con cierto código PHP embebido (introducido) en el mismo, que producirá cierta salida (en nuestro ejemplo, producir un texto). El código PHP se incluye entre etiquetas especiales de comienzo y final que nos permitirán entrar y salir del modo PHP.

Lo que distingue a PHP de la tecnología Javascript, la cual se ejecuta en la máquina cliente, es que el código PHP es ejecutado en el servidor. Si tuviésemos un script similar al de nuestro ejemplo en nuestro servidor, el cliente solamente recibiría el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar que código ha producido el resultado recibido. El servidor web puede ser incluso configurado para que procese todos los ficheros HTML con PHP.

1.2.2 Un poco de Historia sobre PHP.

PHP comenzó siendo un conjunto de scripts escritos en Perl que permitían a su creador, Rasmus Lerdorf , el control de sus accesos a sus páginas personales. A este conjunto de scripts les denomino como *Personal Home Page Tools.* Poco a poco, Ramus fue complementando las funcionalidades básicas de su herramienta escribiendo programas en C. En 1995 decidió liberar el código fuente escrito en C para que cualquiera pudiera utilizarlo e, incluso, colaborar en su mejora. De este modo nació PHP/FI 2.0, pasando de ser el proyecto de una sola persona al desarrollo de un equipo. Tuvo un seguimiento estimado de varios miles de usuarios en todo el mundo, con aproximadamente 500.000 dominios informando que lo tenían instalado, lo que sumaba alrededor del 1% de los dominios de internet.

En junio de 1998 se liberó oficialmente PHP 3.0. Una de las mejores características de PHP 3.0 que atrajo a docenas de desarrolladores a unirse y enviar nuevo módulos de extensión era su gran extensibilidad, además de proveer a los usuarios finales de una sólida infraestructura para muchísimas bases de datos, protocolos y API´s. En su apogeo, PHP 3.0 estaba instalado en aproximadamente un 10% de los servidores Web de Internet.

1.2.3 ¿Qué se puede hacer con PHP?

Aunque principalmente se utiliza para programar scripts que van a ser ejecutados en servidores Web, no hay que olvidar que puede utilizarse como cualquier otro lenguaje para escribir programas que se ejecuten desde la línea de comandos, es decir, sin la necesidad de que se ejecute conjuntamente con un servidor Web. De todas formas, es en el entorno Web donde ha conseguido su mayor aceptación, y es que PHP no solo permite realizar todas las opciones propias sino que también nos proporciona las siguientes posibilidades:

- Soporte para múltiples Sistemas Operativos.
- Soporte para múltiples servidores Web.
- Soporte para más de 25 gestores de bases de datos.
- Soporte para extensiones DBX y ODBC.
- Puede utilizar objetos Java de forma transparente, como objetos PHP.
- PHP soporta intercambio de datos entre lenguajes de programación Web.
- Generación de resultados en múltiples formatos.
- Funciones de comercio electrónico.

1.3 Introducción a Apache.

Apache es un servidor del protocolo http, comúnmente llamado servidor web pues es la mayor utilidad para dicho protocolo. Cuando pensamos en un servidor web imaginamos un grupo de páginas web que determinan un sitio web. Un servidor como Apache puede alojar varios sitios, y pueden coexistir varios servidores Apache en un sólo equipo.

Básicamente Apache lee un directorio con todo el contenido posible a enviar y los navegadores piden las páginas (o recursos) para luego, por ejemplo, mostrarlos en pantalla. Es el funcionamiento más básico de un servidor, sin embargo, los servidores actuales realizan muchas tareas complejas. Un ejemplo sería modificar el recurso para personalizarlo y luego enviarlo. O ejecutar un programa y que la salida de este programa devuelva el recurso a enviar. Comúnmente estos programas se llaman scripts y se tienden a escribir en lenguajes que fueron creados para ese propósito, como lo es PHP, Python o versiones actuales de Perl.

Apache es el servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa.

La historia de Apache se remonta a febrero de 1995, donde empieza el proyecto del grupo Apache, el cual está basado en el servidor Apache httpd de la aplicación original de NCSA. El desarrollo de esta aplicación original se estancó por algún tiempo tras la marcha de Rob McCool por lo que varios webmaster siguieron creando sus parches para sus servidores web hasta que se contactaron vía email para seguir en conjunto el mantenimiento del servidor web, fue ahí cuando formaron el grupo Apache.

Fueron Brian Behlendorf y Cliff Skolnick quienes a través de una lista de correo coordinaron el trabajo y lograron establecer un espacio compartido de libre acceso para los desarrolladores.

Fue así como fue creciendo el grupo Apache, hasta lo que es hoy :) Aquella primera versión y sus sucesivas evoluciones y mejoras alcanzaron una gran implantación como software de servidor inicialmente solo para sistemas operativos UNIX y fruto de esa evolución es la versión para Windows.

Apache es una muestra, al igual que el sistema operativo Linux (un Unix desarrollado inicialmente para PC), de que el trabajo voluntario y cooperativo dentro de Internet es capaz de producir aplicaciones de calidad profesional difíciles de igualar.

La licencia Apache es una descendiente de la licencias BSD, no es GPL. Esta licencia te permite hacer lo que quieras con el código fuente siempre que les reconozcas su trabajo.

Algunas razones por las cuales es tan popular son las siguientes:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Apache es una tecnología gratuita de código fuente abierto. El hecho de ser gratuita es importante pero no tanto como que se trate de código fuente abierto. Esto le da una transparencia a este software de manera que si queremos ver que es lo que estamos instalando como servidor, lo podemos saber, sin ningún secreto, sin ninguna puerta trasera.
- Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades del servidor Web Apache. Actualmente existen muchos módulos para Apache que son adaptables a este, y están ahí para que los instalemos cuando los necesitemos. Otra cosa importante es que cualquiera que posea una experiencia decente en la programación de C o Perl puede escribir un modulo para realizar una función determinada.
- Apache trabaja con gran cantidad de Perl, PHP y otros lenguajes de script. Perl destaca en el mundo del script y Apache utiliza su parte del pastel de Perl tanto con soporte CGI como con soporte mod perl. También trabaja con Java y páginas jsp. Teniendo todo el soporte que se necesita para tener páginas dinámicas.
- Apache te permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.
- Tiene una alta configurabilidad en la creación y gestión de logs. Apache permite la creación de ficheros de log a medida del administrador, de este modo puedes tener un mayor control sobre lo que sucede en tu servidor.
- Se pueden extender las características de Apache hasta donde nuestra imaginación y conocimientos lleguen.

1.4 Introducción a MySQL.

MySQL es un sistema gestor de bases de datos relacionales en SQL, esto significa que permite la gestión de los datos de una BD relacional usando un lenguaje de consulta estructurado. Y, por tanto, que a partir de una oración, MySQL llevará a cabo una determinada acción sobre nuestra base de datos. MySQL es una aplicación de Código abierto y por lo tanto es gratuita, nos permite redistribuir una aplicación que la contenga y nos permite incluso modificar su código para mejorarla o adaptarla a nuestras necesidades.

Además, existe la seguridad de contar con una importante cuota de mercado y de saber que es una solución estable, mantenida por un buen equipo de desarrolladores y e incluso con soporte de pago. MySQL es un sistema fácil de instalar y configurar en servidores Windows, Linux. En la funcionalidad quizás MySQL flaquea un poco frente a sus rivales, pero sin embargo dispone de muchas funciones vitales para el desarrollo profesional cómo puede ser el volcado online, la duplicación… etc.

MySQL puede correr en la inmensa mayoría de sistemas operativos, por lo que junto a otro lenguaje de programación de lado de servidor de alta portabilidad como Java, PHP, Perl... nos permite el desarrollo de aplicaciones web fáciles de migrar y el acceso y copia de los datos desde cualquier Sistema Operativo.

2.- Herramientas y Tecnologías a Utilizar.

Es necesario explicar cuáles y para qué sirven las herramientas a utilizar con la intención de conocer los atributos y en caso de ser necesario buscar alternativas con algún otro programa mejor o más actual.

2.1 Xampp.

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor Web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, MySQL, PHP, Perl. El programa está liberado bajo la licencia GNU y actúa como un servidor Web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris, y MacOS X.

XAMPP solamente requiere descargar y ejecutar un archivo zip, tar, o exe, con unas pequeñas configuraciones en alguno de sus componentes que el servidor Web necesitará. XAMPP se actualiza regularmente para incorporar las últimas versiones de Apache/MySQL/PHP y Perl. También incluye otros módulos como OpenSSL y phpMyAdmin. Para instalar XAMPP se requiere solamente una pequeña fracción del tiempo necesario para descargar y configurar los programas por separado.

Oficialmente, los diseñadores de XAMPP sólo pretendían su uso como una herramienta de desarrollo, para permitir a los diseñadores de sitios webs y programadores testear su trabajo en sus propios ordenadores sin ningún acceso a Internet. En la práctica, sin embargo, XAMPP es utilizado actualmente para servidor de sitios Web y, con algunas modificaciones, es generalmente lo suficientemente seguro para serlo. Con el paquete se incluye una herramienta especial para proteger fácilmente las partes más importantes.

2.2 EMS MySQL.

EMS MySQL Manager proporciona un completo conjunto de eficaces y potentes herramientas para administrar un servidor MySQL.

A través de su clara interfaz gráfica permite crear y editar parámetros de una base de datos de forma sencilla. Ofrece la posibilidad de otorgar y administrar privilegios de usuarios, ejecutar scripts SQL, queries visuales integradas, extraer o imprimir metadata, importar y exportar datos, etc

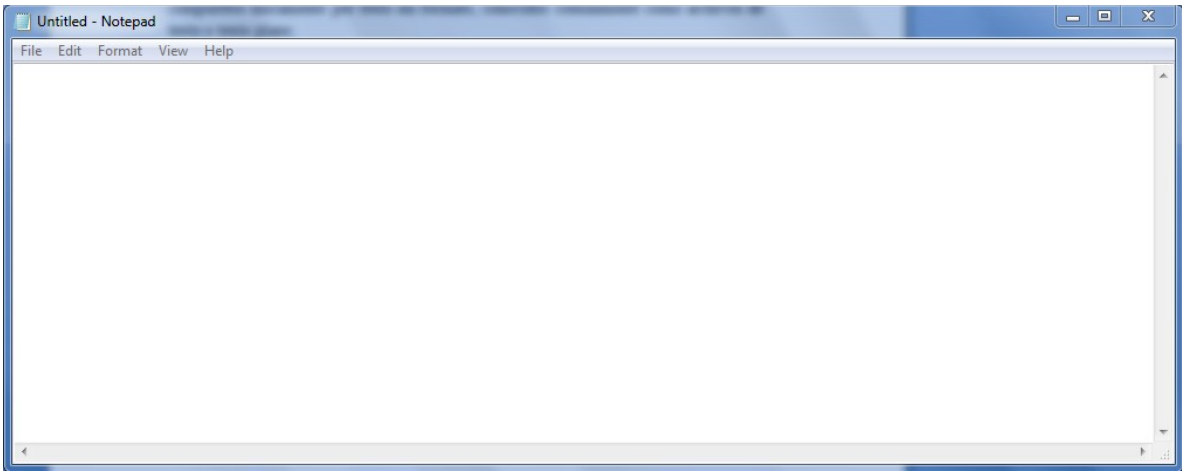
2.3 Editor de Texto.

Un editor de texto es un programa que permite crear y modificar archivos digitales compuestos únicamente por texto sin formato, conocidos comúnmente como archivos de texto o texto plano.

Hay una gran variedad de editores de texto. Algunos son de uso general, mientras que otros están diseñados para escribir o programar en un lenguaje. Algunos son muy sencillos, mientras que otros tienen implementadas gran cantidad de funciones.

El editor de texto debe ser considerado como una herramienta de trabajo del programador o administrador de la máquina. Como herramienta permite realizar ciertos trabajos, pero también requiere de aprendizaje para que el usuario conozca y obtenga destreza en su uso. La llamada curva de aprendizaje es una representación de la destreza adquirida a lo largo del tiempo de aprendizaje. Un editor puede ofrecer muchas funciones, pero si su curva de aprendizaje es muy larga, puede desanimar el aprendizaje y terminará siendo dejado de lado. Puede que un editor tenga una curva de aprendizaje muy empinada y corta, pero si no ofrece muchas funciones el usuario le reemplazará por otro más productivo. Es decir la elección del editor más apropiado depende de varios factores, alguno de ellos muy subjetivos. Esta coyuntura de intereses ha dado lugar a largas discusiones sobre la respuesta a la pregunta "¿cuál es el mejor editor de texto?". Hoy en día muchos editores originalmente salidos de Unix o Linux han sido portados a otros sistemas operativos, lo que permite trabajar en otros sistemas sin tener que aprender el uso de otro editor.

Algunos editores son sencillos mientras que otros ofrecen una amplia gama de funciones.



3.- Instalación de Aplicaciones.

A continuación mostraremos la forma correcta de instalar las herramientas para su uso durante el curso.

3.1. Instalación del Xampp.

Evidentemente lo primero que tenemos que hacer el irnos a la página oficial de XAMPP y bajarnos el instalador. Para este tutorial vamos a instalar XAMPP en una máquina con Windows XP Pro y vamos a usar el instalador (http://www.apachefriends.org/en/xampp-windows.html). También se puede instalar sin instalador descomprimiendo el ZIP directamente en nuestra máquina.

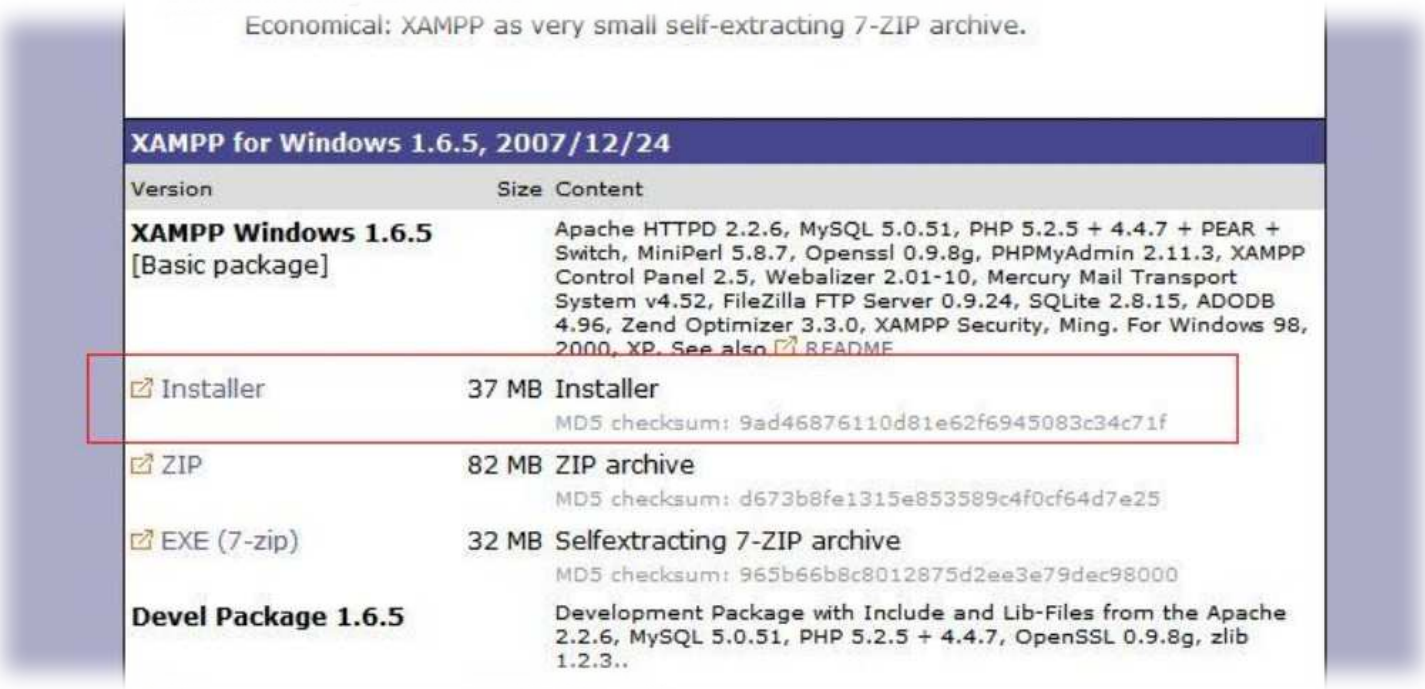


Imagen 3.1. Localizando el instalador en la página oficial del Xampp.

Una vez descargado el instalador vamos a empezar a instalar ....

Imagen 3.2. Opción para seleccionar el idioma del programa.

Elegimos el idioma y pulsamos OK

Imagen 3.3. Pantalla principal del instalador.

Pulsamos el botón Next.

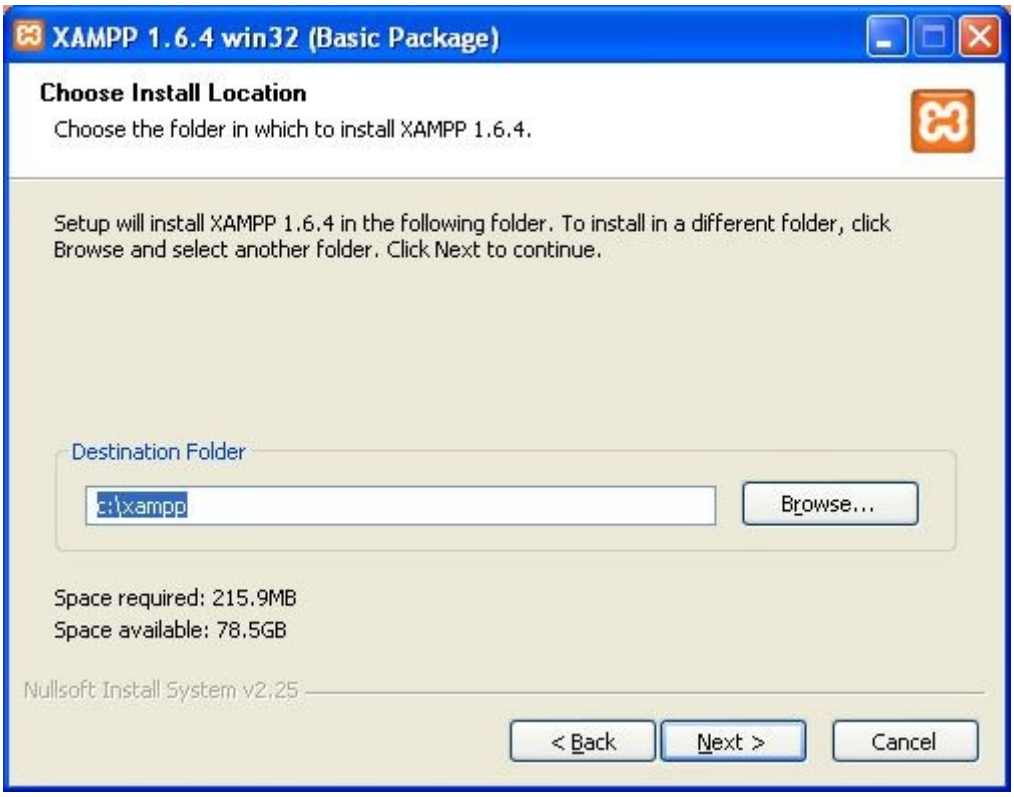


Imagen 3.4. Ruta a instalar el Xampp.

Seleccionamos la carpeta destino donde se instalará la herramienta.

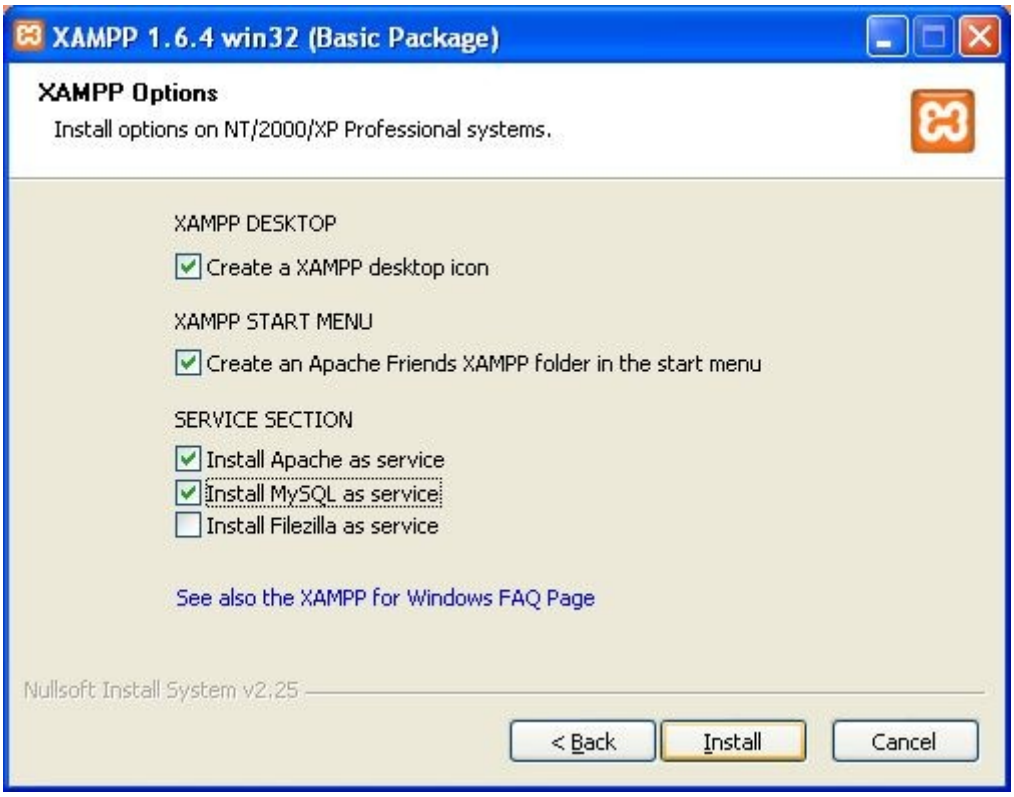


Imagen 3.5. Opciones a instalar en el Xampp.

En este paso podemos elegir instalar los distintos módulos como servicios de Windows. Para el ejemplo sólo vamos a hacerlo para el Apache y MySQL.

Imagen 3.6. Pantalla final del instalador.

Cuando la instalación termine hacemos click en el botón Finish.

Administración del Xampp.

Una vez instalado correctamente XAMPP vamos a ver las posibles opciones de configuración y administración de la herramienta y sus módulos instalados, para ello arrancamos el panel de control de XAMPP.

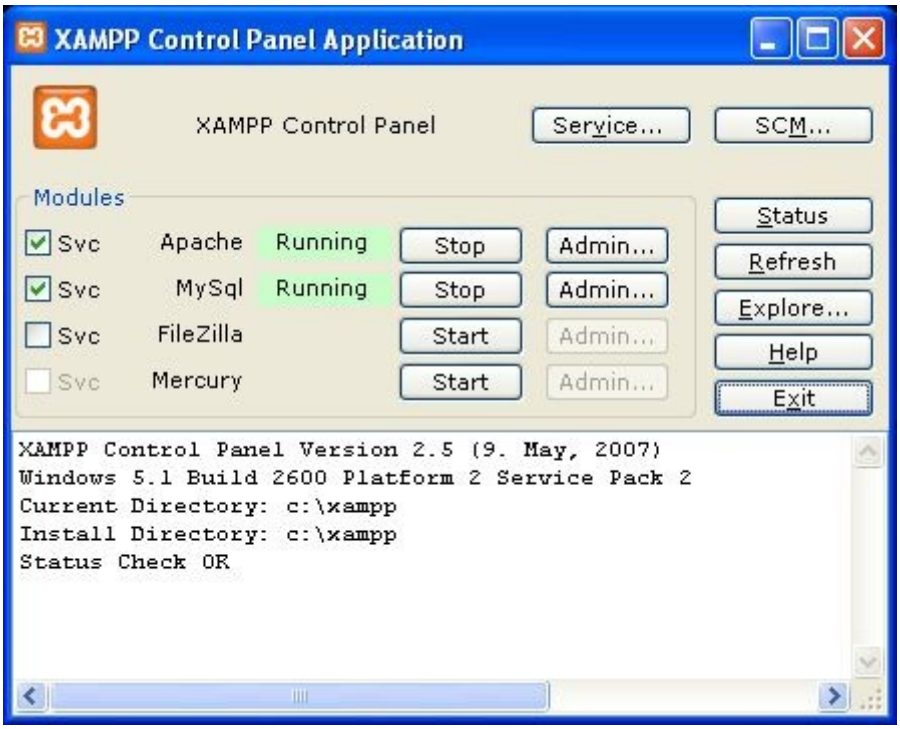


Imagen 3.7. Pantalla de Administración del Xampp.

En este panel de control podemos ver todos los módulos instalados. Para cada módulo podremos parar su servicio (*Stop*), arrancarlo (*Start*), ver su estado (*Stop / Running*), marcarlo como servicio (*checkbox Svc*) y entrar en su panel de administración (*Admin*).

Para probar que la instalación de XAMPP fue exitosa basta con poner en el navegador "**http://localhost**"o "**http://127.0.0.1**"y nos aparecerá la aplicación de administración web. En ella tenemos un sección de administración web de XAMPP, una sección de insteresante demos y otra con herramientas incluidas en el paquete como *phpMyAdmin, FileZilla FTP, Webalizer*, etc.

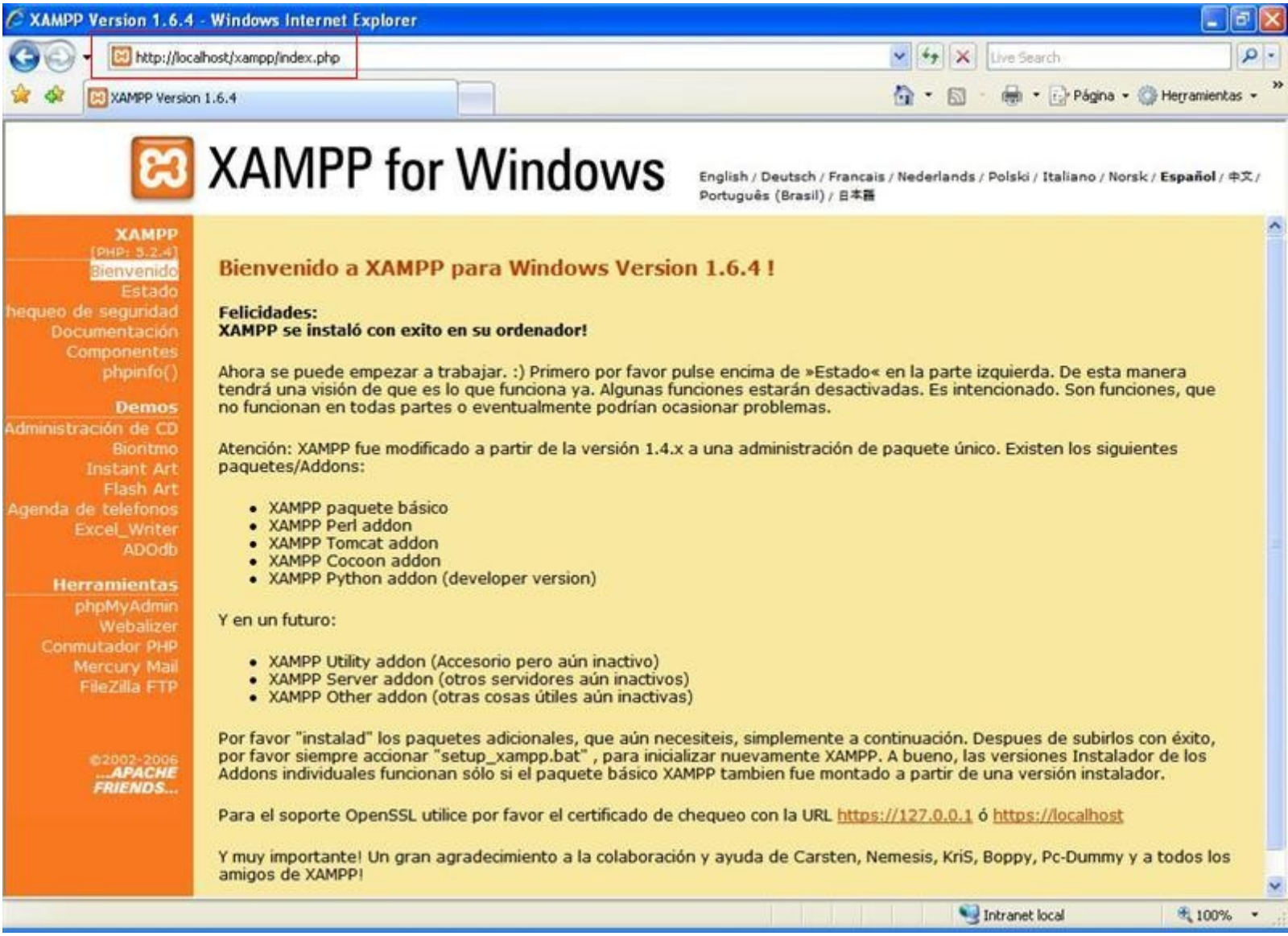


Imagen 3.8. Pantalla Principal del Xampp.

En la primera sección podremos ver toda la información relativa al PHP instalado (*phpinfo()*). Además tenemos toda la documentación online de cada uno de los componentes y módulos incorporados en la herramienta. Como se dijo anteriormente XAMPP se desarrolló inicialmente para entornos de desarrollo y no para entornos de producción, como se puede ver en la opción de chequear seguridad hay muchos agujeros y deficiencias. Para solucionar estos agujeros vaya a este enlace "**http://localhost/security/xamppsecurity.php**" antes de publicar nada en su website.

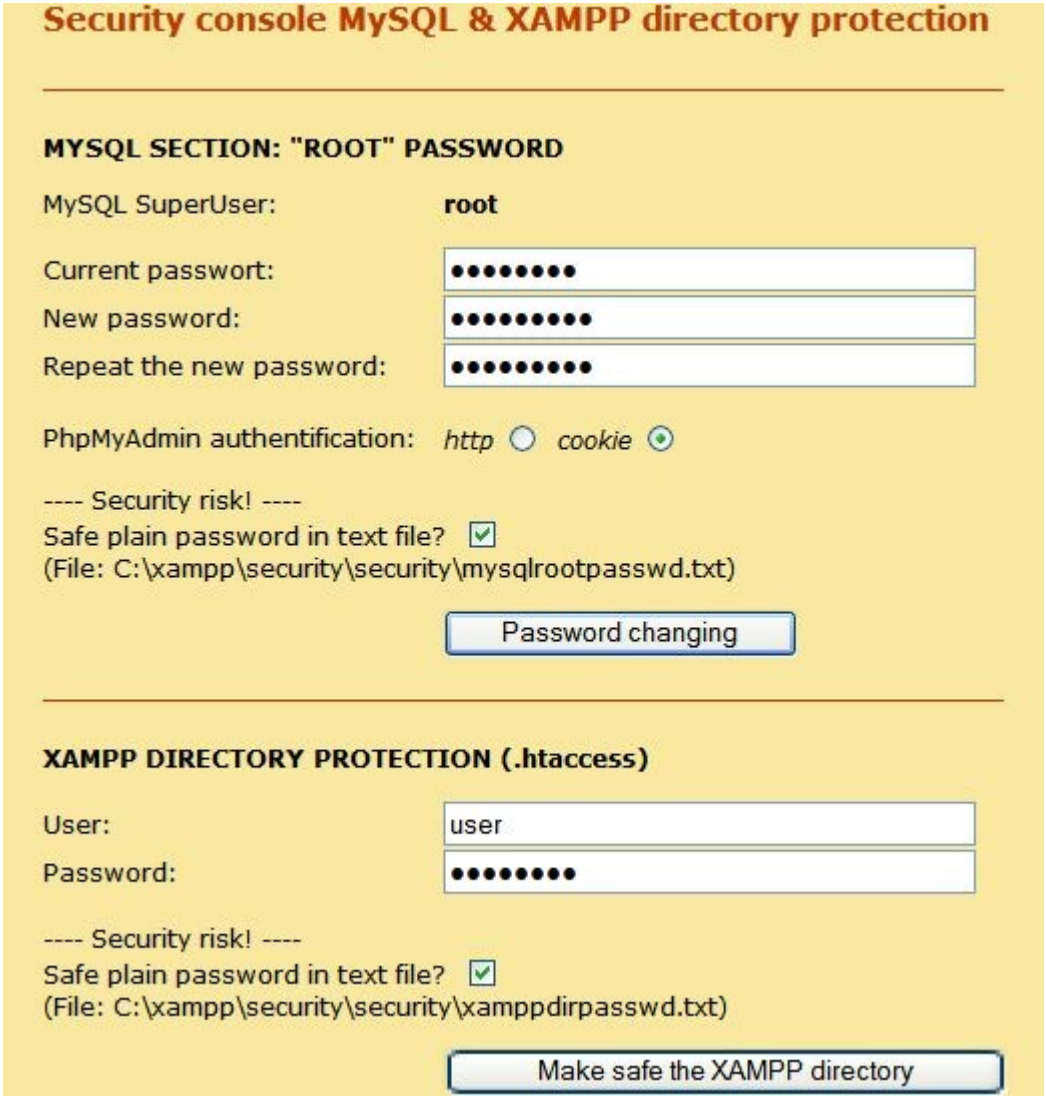


Imagen 3.9. Pantalla de configuración de seguridad.

En la sección de herramientas hay dos bastante interesante, *phpMyAdmin* y *Conmutador PHP*. La primera de ellas, phpMyAdmin es una herramienta muy conocida que nos permitirá administrar nuestro MySQL. Antes de empezar a trabajar con esta herramienta hay que configurar algunas cosillas.

Inicialmente *MySQL* crea un usuario por defecto llamado *root* sin password. Para poder cambiar la password de *root* se debe acceder a la administración de MySQL a través del Panel de Control de XAMPP. Después de guardar los cambios, hay que modificar el fichero "**config.inc.php**" situado en *"$HOME_XAMPP/phpMyAdmin/"* y editar las siguientes líneas:

```
$cfg['Servers'][$i]['auth_type']     = 'config';
$cfg['Servers'][$i]['user']          = 'root';
$cfg['Servers'][$i]['password']      = 'XXXXXX'
```

En este fichero se configurarán las variables necesarias para que phpMyAdmin pueda acceder a MySQL, las más importantes son *auth_type* para el tipo de autenticación, *user* y *password*. Para la variable *auth_type* podemos poner el método de autenticación **http** y cuando accedamos a phpMyAdmin nos aparecerá una ventana para introducir el usuario y password de MySQL. Sin embargo, si ponemos como método de autenticación **config** debemos poner en las variables *user* y *password* el usuario y password de MySQL y de esta forma accederá directamente a phpMyAdmin sin preguntar nada el usuario.

Imagen 3.10. Pantalla de configuración de PHP MyAdmin.

Otra herramienta interesante es el conmutador de PHP, es decir, una forma rápida de cambiar la configuración de PHP de PHP 4 a PHP 5 y viceversa. Para poder conmutar el PHP antes tenemos que parar el Apache. Después de esto sólo tenemos que ejecutar el script "**php-switch.bat**" situado en la carpeta principal de la instalación de XAMPP.

```
  Starting configure XAMPP with PHP 4 ...

  Installing PHP4 in XAMPP now!

  Copy the current php.ini to C:\xampp\php\php5.ini ... done!
  Copy the php4.ini to C:\xampp\apache\bin\php.ini ... done!

  Change PHP settings in C:\xampp\apache\conf\extra\httpd-xampp.conf ... done!

  Copy now all php4 dlls to C:\xampp\apache\bin

C:\xampp\php\php4\expat.dll => C:\xampp\apache\bin\expat.dll
C:\xampp\php\php4\FDFTK.DLL => C:\xampp\apache\bin\FDFTK.DLL
C:\xampp\php\php4\freetype6.dll => C:\xampp\apache\bin\freetype6.dll
C:\xampp\php\php4\fribidi.dll => C:\xampp\apache\bin\fribidi.dll
C:\xampp\php\php4\gds32.dll => C:\xampp\apache\bin\gds32.dll
C:\xampp\php\php4\iconv.dll => C:\xampp\apache\bin\iconv.dll
C:\xampp\php\php4\jpeg62.dll => C:\xampp\apache\bin\jpeg62.dll
C:\xampp\php\php4\libeay32.dll => C:\xampp\apache\bin\libeay32.dll
C:\xampp\php\php4\libexpat.dll => C:\xampp\apache\bin\libexpat.dll
C:\xampp\php\php4\libmcrypt.dll => C:\xampp\apache\bin\libmcrypt.dll
C:\xampp\php\php4\libmhash.dll => C:\xampp\apache\bin\libmhash.dll
C:\xampp\php\php4\libmysql.dll => C:\xampp\apache\bin\libmysql.dll
C:\xampp\php\php4\libpng12.dll => C:\xampp\apache\bin\libpng12.dll
C:\xampp\php\php4\mSQL.dll => C:\xampp\apache\bin\mSQL.dll
C:\xampp\php\php4\msvcr71.dll => C:\xampp\apache\bin\msvcr71.dll
C:\xampp\php\php4\ntwdblib.dll => C:\xampp\apache\bin\ntwdblib.dll
C:\xampp\php\php4\php4apache2.dll => C:\xampp\apache\bin\php4apache2.dll
C:\xampp\php\php4\php4ts.dll => C:\xampp\apache\bin\php4ts.dll
C:\xampp\php\php4\phpsrvlt.jar => C:\xampp\apache\bin\phpsrvlt.jar
C:\xampp\php\php4\php_java.jar => C:\xampp\apache\bin\php_java.jar
C:\xampp\php\php4\sablot.dll => C:\xampp\apache\bin\sablot.dll
C:\xampp\php\php4\ssleay32.dll => C:\xampp\apache\bin\ssleay32.dll
C:\xampp\php\php4\Yaz.dll => C:\xampp\apache\bin\Yaz.dll
C:\xampp\php\php4\zlib1.dll => C:\xampp\apache\bin\zlib1.dll
  Write the new PHP main version in C:\xampp\install\.phpversion

  OKAY ... PHP SWITCHING WAS SUCCESSFUL

  Now you can start the Apache with PHP 4 !
  Nun kannst du den Apache mit PHP 4 starten!

  :-) Kay Vogelgesang & Carsten Wiedmann (www.apachefriends.org)

Presione una tecla para continuar . . .
```

Imagen 3.11. Instalación de PHP en el Xampp.

Después de ejecutar el script vamos a comprobar que la conmutación se ha realizado correctamente.

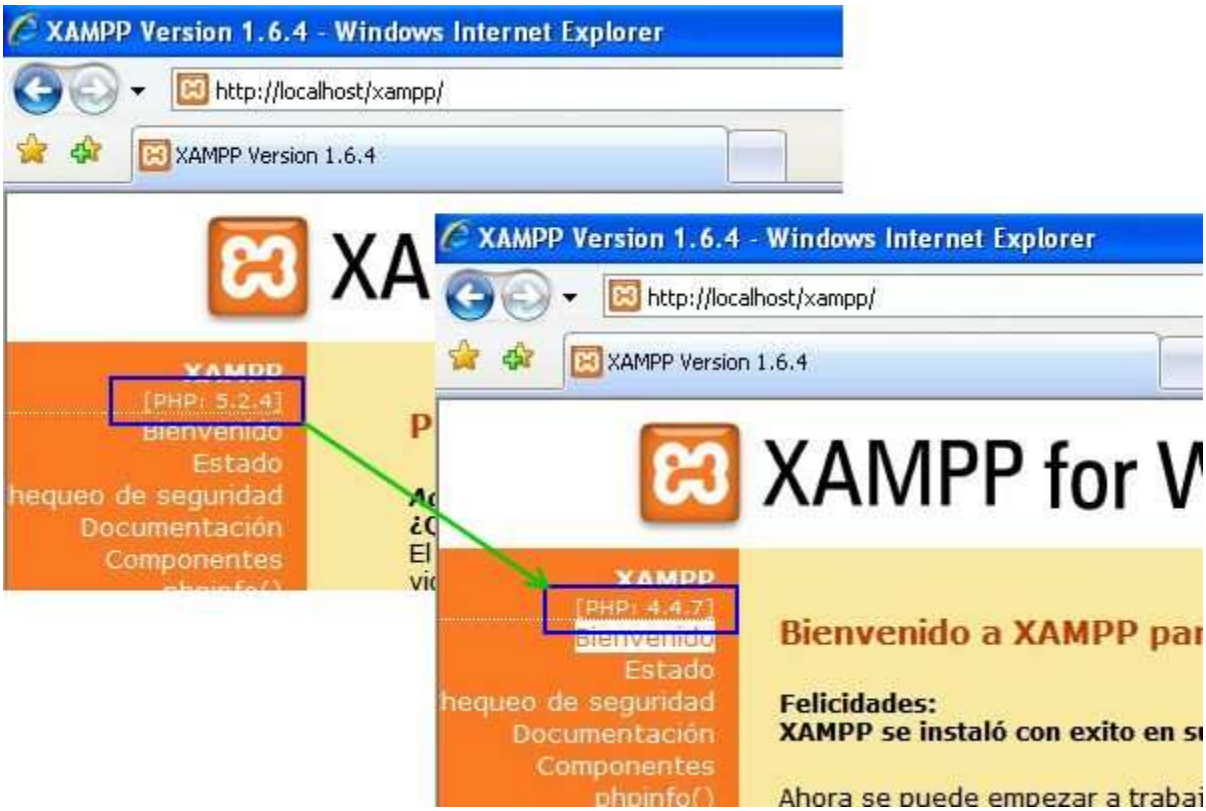


Imagen 3.12. Pantalla de verificación de la configuración del Xampp.

Ahora vamos a probar con un sencillo ejemplo. Este ejemplo está realizado en PHP, se conecta a base de datos y recupera datos de una determinada tabla. El resultado es este:

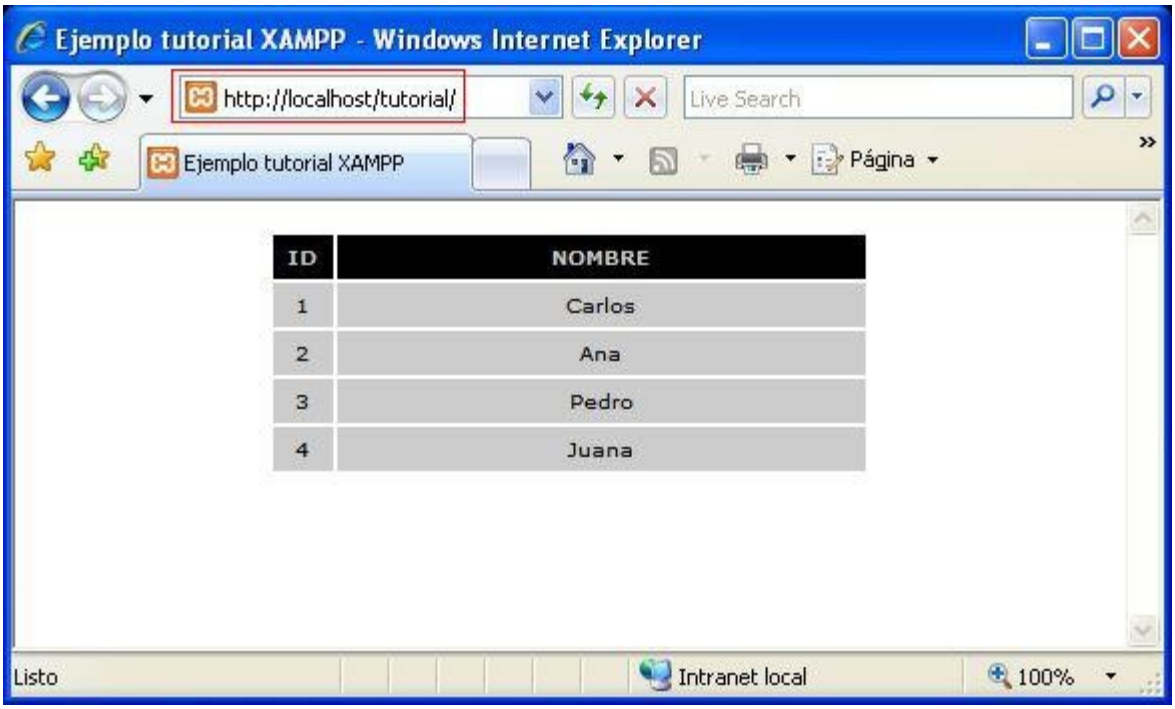


Imagen 3.13. Ejemplo de conexión con Xampp.

Pero, ¿dónde tenemos que dejar nuestras aplicaciones web para poder ejecutarlas con xampp?

Hay que dejarlas dentro de la carpeta "**htdocs**" situada en la carpeta principal de la instalación de XAMPP. Para nuestro ejemplo hemos creado una nueva carpeta llamada "tutorial" y hemos colocado ahí el ejemplo.

Imagen 3.14. Pantalla que muestra la ubicación de la carpeta htdocs.

Para poder ver las aplicaciones creadas basta con introducir en la barra de direcciones del navegador el path relativo a partir de la carpeta "htdocs" justo despúes de "http://localhost" (en el ejemplo nuestro "**http://localhost/tutorial**").

3.2. Instalación de EMS MySQL Manager.

Veamos cómo es la instalación de la herramienta MySQL Manager.

Primeramente ejecutamos el instalador de la aplicación.

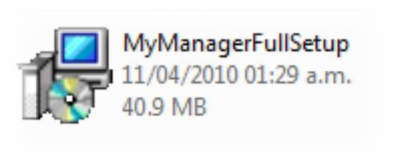


Imagen 3.15 Instalador de MySQL Manager.

Seleccionamos el idioma con el cual instalaremos la aplicación y damos clic en Next. En esta ocasión seleccionaremos el idioma ingles debido a que esta versión solo cuenta con 3 idiomas.

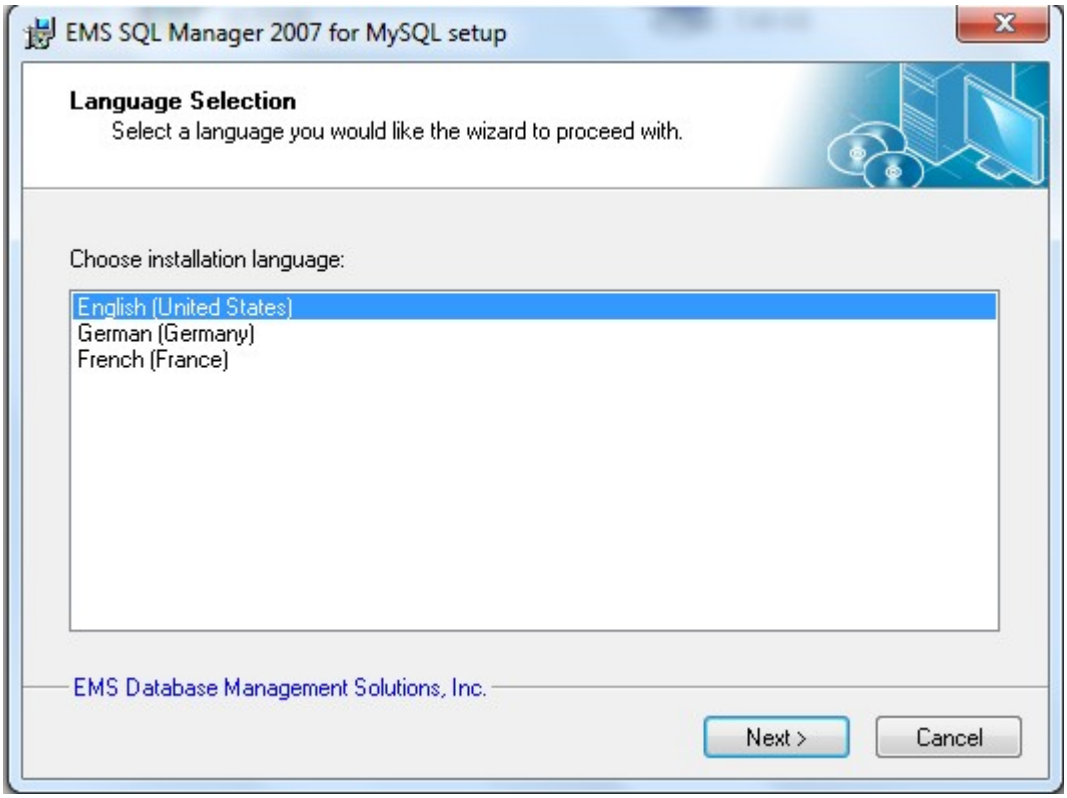


Imagen 3.16. Pantalla de selección de idioma.

Esta es la pantalla principal del instalador, para continuar damos clic en Next.

Imagen 3.17. Pantalla principal del instalador.

La siguiente pantalla es donde aceptamos los términos de la licencia. Seleccionamos la casilla de YES y damos clic en el botón Next.

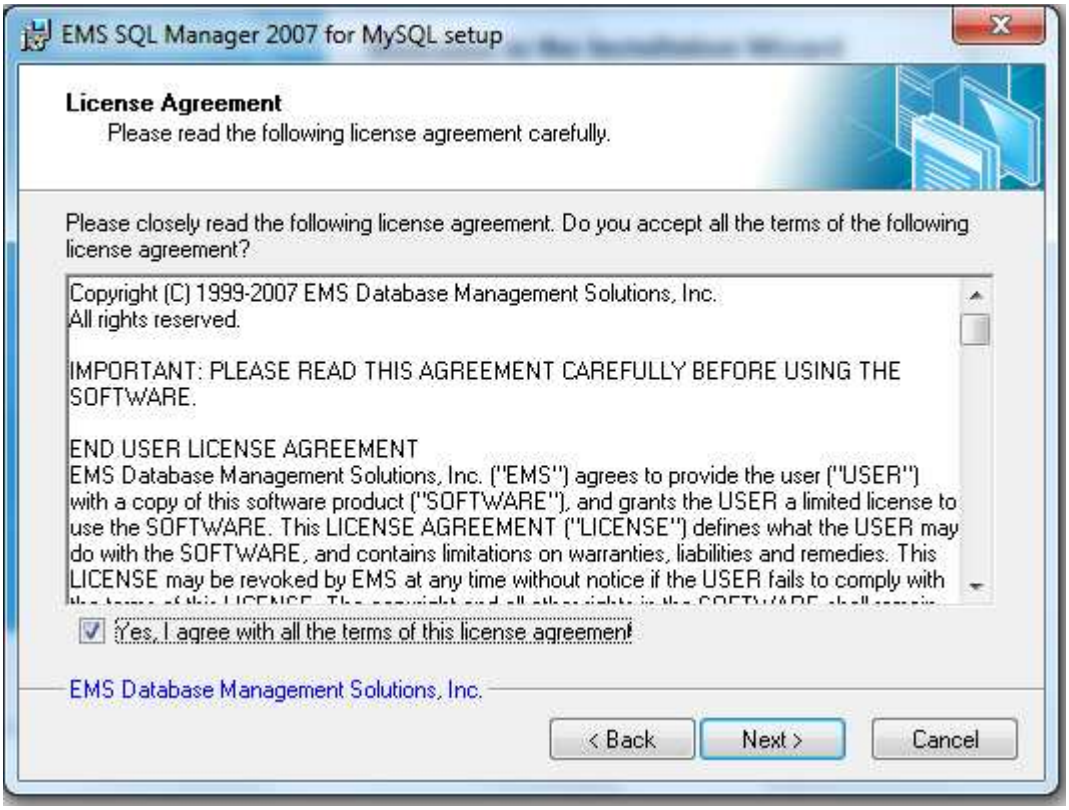


Imagen 3.18. Pantalla de aceptación de los términos de la licencia.

La siguiente pantalla es donde elegimos la ruta de instalación, dejamos la ruta que aparece por default y damos clic en Next.

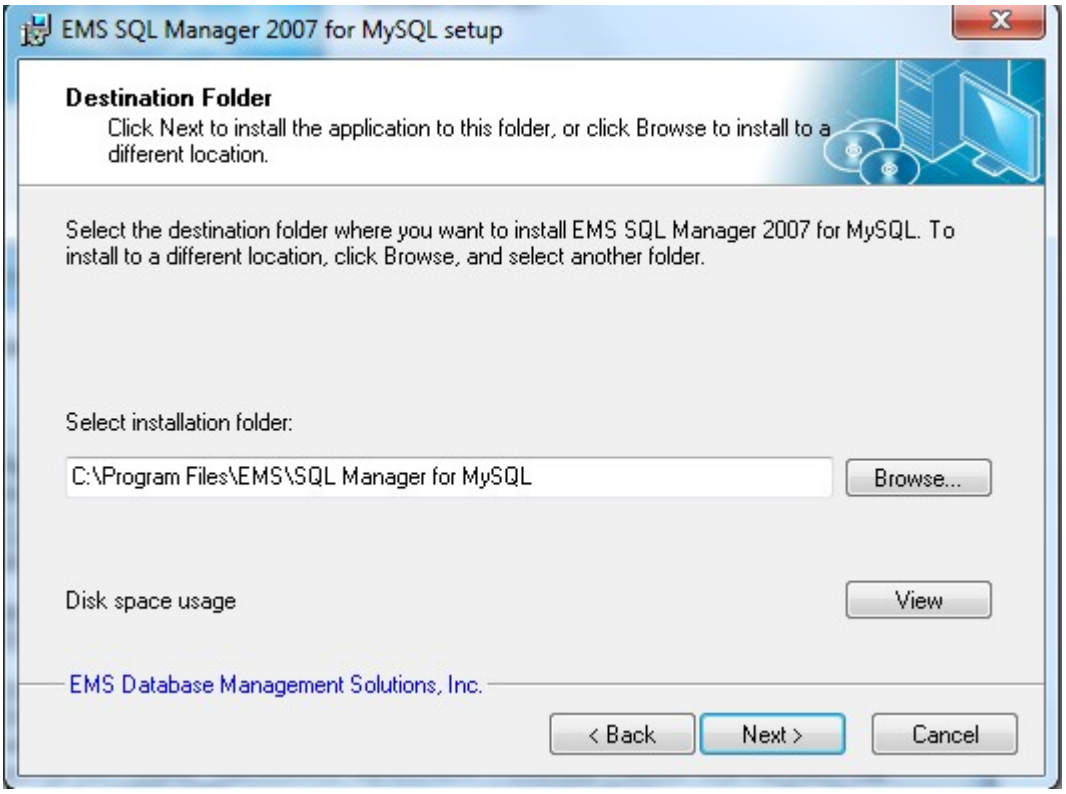
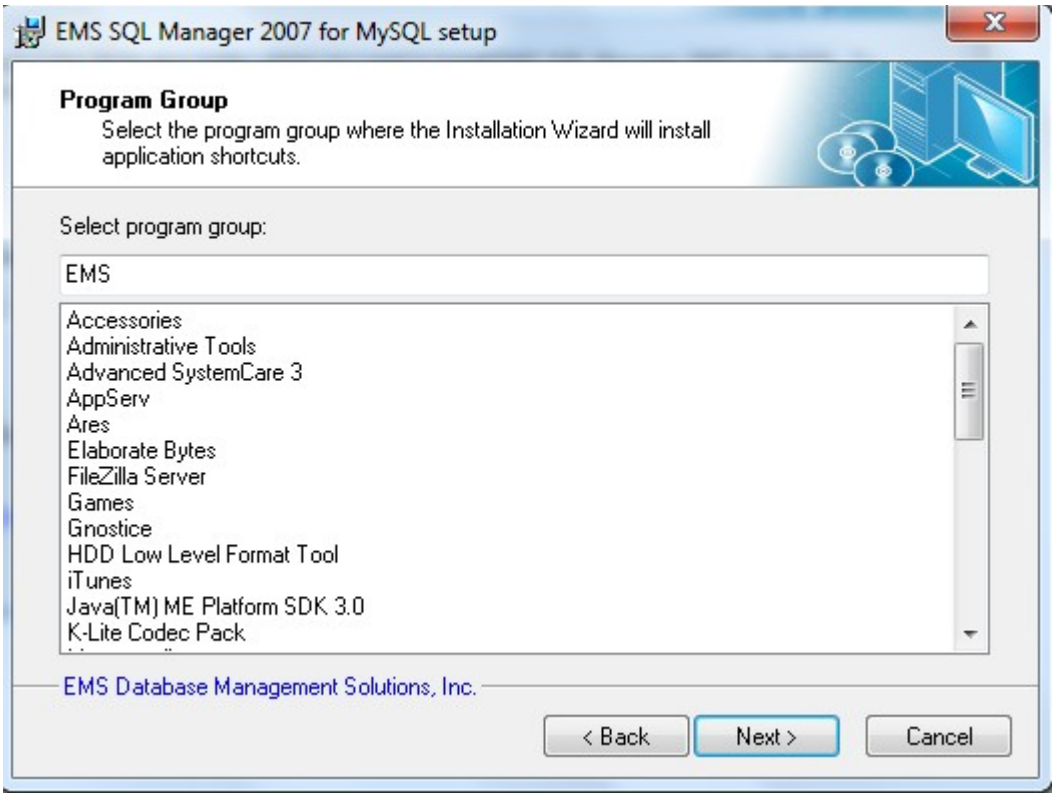


Imagen 3.19. Pantalla de elección de ruta de instalación.

La siguiente pantalla nos muestra el grupo donde se va a realizar la instalación. Damos clic en Next para continuar.



3.20. Pantalla donde se muestra el grupo de programas donde se realizara la instalación.

La siguiente pantalla es un previo a la instalación, para continuar hay que dar clic en Next.

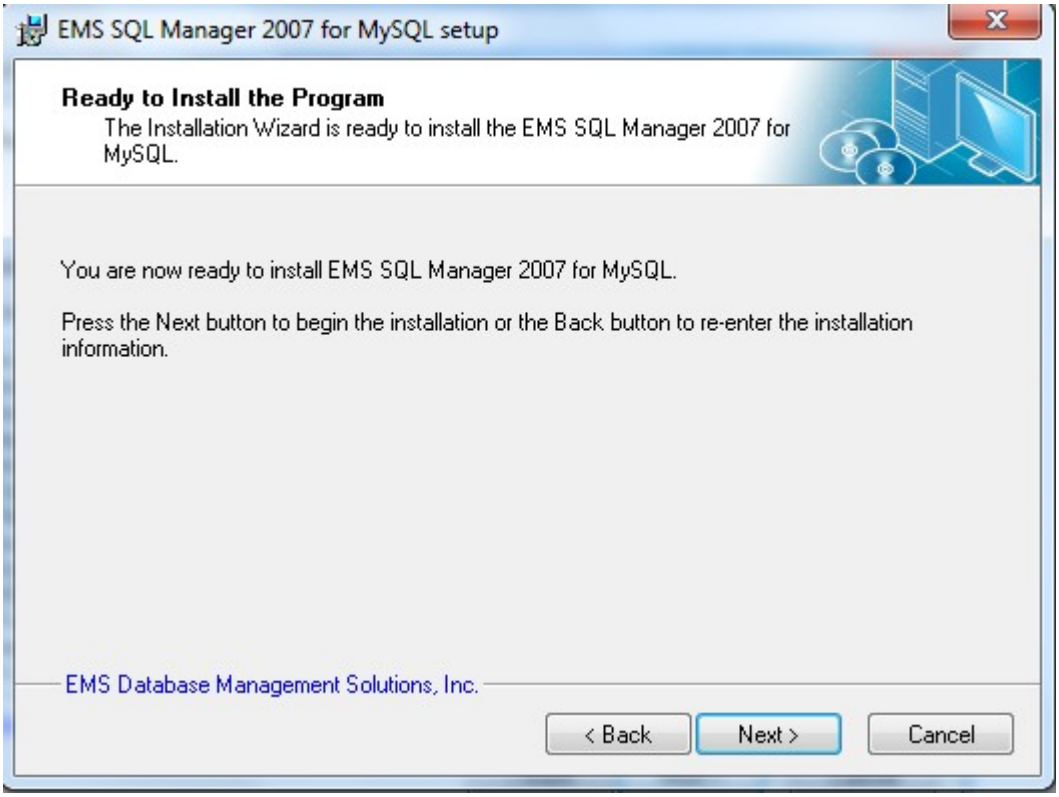


Imagen 3.21. Pantalla previa a la instalación.

La siguiente pantalla es la mostrada cuando la instalación termino correctamente, deshabilitamos la primera opción de ejecutar el programa en este momento y damos clic en finish.

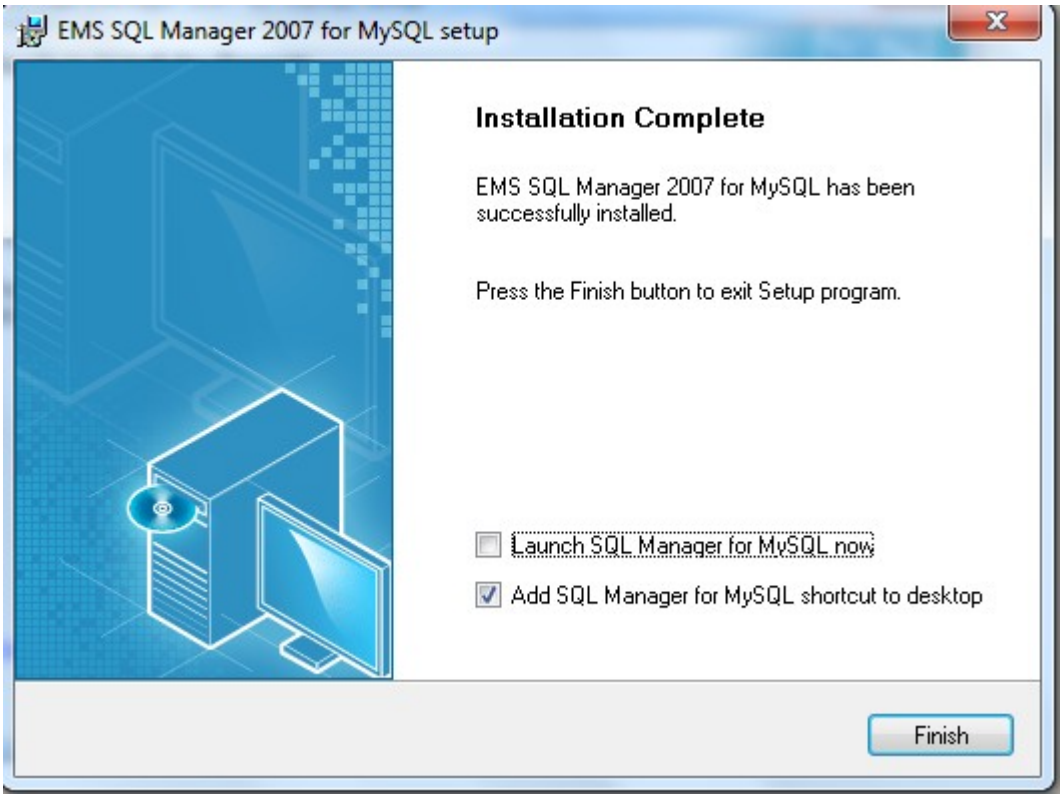


Imagen 3.22. Pantalla final de la instalación.

Una vez instalada la aplicación es necesario instalar el parche para no tener problemas posteriormente.

Ejecutaremos primero el archivo llamado “ inv ” , la cual es una entrada del registro, al hacer esto nos mandara una advertencia en la cual solo tendremos que dar clic en aceptar.

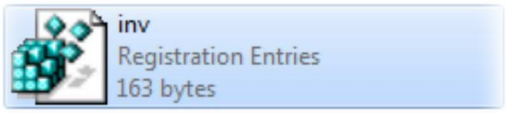


Imagen 3.23. Archivo de entrada del registro.

Ahora copiaremos el archivo llamado MyManager en la misma ubicación donde instalamos el manejador. Al mostrar la advertencia de que tenemos un archivo con el mismo nombre en esa ubicación le damos sobrescribir y después damos clic en continuar. Con esto nuestro manejador quedara listo para su utilizarlo.

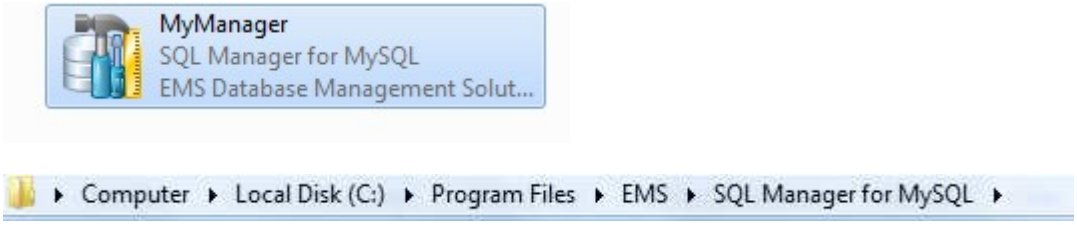


Imagen 3.24. Pantalla donde se muestra el archivo necesario a copiar y su ruta.

4.- HTML

HTML, siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo Javascript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

HTML es el lenguaje con el que se escriben las páginas web. Las páginas web pueden ser vistas por el usuario mediante un tipo de aplicación llamada navegador. Podemos decir por lo tanto que el HTML es el lenguaje usado por los navegadores para mostrar las páginas webs al usuario, siendo hoy en día la interface más extendida en la red.

Este lenguaje nos permite aglutinar textos, sonidos e imágenes y combinarlos a nuestro gusto. Además, y es aquí donde reside su ventaja con respecto a libros o revistas, el HTML nos permite la introducción de referencias a otras páginas por medio de los enlaces hipertexto.

El HTML se creó en un principio con objetivos divulgativos. No se pensó que la web llegara a ser un área de ocio con carácter multimedia, de modo que, el HTML se creó sin dar respuesta a todos los posibles usos que se le iba a dar y a todos los colectivos de gente que lo utilizarían en un futuro. Sin embargo, pese a esta deficiente planificación, si que se han ido incorporando modificaciones con el tiempo, estos son los estándares del HTML. Numerosos estándares se han presentado ya. El HTML 4.01 es el último estándar a septiembre de 2001.

Esta evolución tan anárquica del HTML ha supuesto toda una seria de inconvenientes y deficiencias que han debido ser superados con la introducción de otras tecnologías accesorias capaces de organizar, optimizar y automatizar el funcionamiento de las webs. Ejemplos que pueden sonaros son las CSS, JavaScript u otros. Veremos más adelante en qué consisten algunas de ellas.

Otros de los problemas que han acompañado al HTML es la diversidad de navegadores presentes en el mercado los cuales no son capaces de interpretar un mismo código de una manera unificada. Esto obliga al programador a, una vez creada su página, comprobar que esta puede ser leída satisfactoriamente por todos los navegadores, o al menos, los más utilizados.

Además del navegador necesario para ver los resultados de nuestro trabajo, necesitamos evidentemente otra herramienta capaz de crear la página en sí. Un archivo HTML (una página) no es más que un texto. Es por ello que para programar en HTML necesitamos un editor de textos.

Es recomendable usar el Bloc de notas que viene con windows, u otro editor de textos sencillo. Hay que tener cuidado con algunos editores más complejos como Wordpad o Microsoft Word, pues colocan su propio código especial al guardar las páginas y HTML es únicamente texto plano, con lo que podremos tener problemas.

Existen otro tipo de editores específicos para la creación de páginas web los cuales ofrecen muchas facilidades que nos permiten aumentar nuestra productividad. No obstante, es aconsejable en un principio utilizar una herramienta lo más sencilla posible para poder prestar la máxima atención a nuestro código y familiarizarnos lo antes posible con él. Siempre tendremos tiempo más delante de pasarnos a editores más versátiles con la consiguiente ganancia de tiempo.

Así pues, una página es un archivo donde está contenido el código HTML en forma de texto. Estos archivos tienen extensión .html o .htm (es indiferente cuál utilizar). De modo que cuando programemos en HTML lo haremos con un editor de textos y guardaremos nuestros trabajos con extensión .html, por ejemplo mipágina.html.

4.1 Etiquetas Básicas.

Las etiquetas básicas son aquellas etiquetas que siempre aparecen en la parte superior de toda página web. No todas estas etiquetas son esenciales para la visualización, aunque siempre es recomendable colocarlas para que los buscadores y los navegadores saquen provecho de ellas.
Una página HTML comienza y termina con las respectivas etiquetas de apertura **<HTML>** y **</HTML>**. Luego, se divide en dos partes: el encabezado entre las etiquetas **<HEAD>** y **</HEAD>** y el cuerpo de la pagina entre las etiquetas **<BODY>** y **</BODY>**. La etiqueta **<HTML>** también puede tener el atributo **lang,** con el cual se especifica el idioma; por ejemplo **<HTML lang = "es">** especifica que la pagina esta en español.

Dentro del encabezado de la pagina se coloca la información del título. Conjunto de caracteres, meta-tags, etc. Una página típica tendría una forma como la que se muestra en la imagen 4.1.

File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>

                <title>Titulo de la pagina</title>

                <meta name= "descripcion" value = "Esta es una pagina de prueba" />
                <meta name= "autor" value= "topico 2" />
                <meta name="keywords" value=" pruebas, pagina, html"/>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

                <LINK type= "text/css" href="varios/estilos.css">

        </head>

        <body>

                <!Contenido del cuerpo de la pagina-->

        </body>

</html>
```

Imagen 4.1 Ejemplo de HTML.

Se puede notar que la primera línea se encuentra fuera de todas las edmas etiquetas. Se trata de la definición del tipo de documento (HTML) y de la versión del lenguaje utilizado. Ademas, encontramos tres nuevas etiquetas, todas dentro del encabezado.

La etiqueta **<TITLE>** permite la definición del título que aparecerá en la parte superior del navegador. Si no se coloca, parecerá la URL de la página, al menos en la mayor parte de los navegadores conocidos.

La etiqueta **<META>** tiene usos múltiples. En los 3 primeros casos la utilizamos para especificar la descripción del autor y las palabras clave de la página (descripción, autor, keywords). Estos datos (keywords) son indexados por los buscadores para luego poder encontrar nuestra página en caso de que se realice alguna consulta. En el cuarto uso de la etiqueta **<META>**, estamos especificando el conjunto de caracteres utilizados en la página.

Por último, la etiqueta **<LINK>** permite vincular un archivo externo a la página. En el ejemplo estamos vinculando un archivo de hoja de estilos.

4.2 El cuerpo de una página.

La etiqueta <**BODY**> tiene una serie de atributos que permiten definir propiedades de la pagina tales como el color de fondo, el color del texto y los links, entre otras cosas, aunque desde la versión HTML 4.0 han quedado obsoletos y reemplazados por las hojas de estilo sin embargo los navegadores los siguen utilizando y es bueno conocerlos. Un ejemplo del uso de colores en una página seria:

Ejemplo: <BODY bgcolor = ”yellow” text = “black” link = “blue” vlink = ,”green” alink = “red”>

El valor **bgcolor** corresponde al color de fondo, mientras que **text** y **link** corresponden al color del texto general y de los hipervínculos respectivamente. Los parámetros **vlink** y alink se refieren a los colores de los hipervínculos ya visitados y los activos.

4.3 Presentación del texto.

Un punto importante es que cuando ingresemos texto en una página Web (o casi cualquier otro elemento) hay que tener en cuenta que no siempre se va a ver en el navegador de quien visite ese documento de la misma manera que lo vemos nosotros. Esto se debe a muchos factores, entre ellos el navegador utilizado, las fuentes instaladas, la resolución de pantalla y el tamaño de la pantalla que esté utilizando. Nosotros debemos hacer que nuestra página se vea lo mejor posible en todos los casos comentados.

Si comenzamos con escribir texto directamente en la sección <**BODY>** y abrimos la pagina en el navegador**,** veremos que mostrará este texto aunque con algunas particularidades.

Si nosotros ingresamos retornos de línea con el texto, estos serán interpretados como espacios en blanco en el navegador.

También, si ingresamos varios espacios en blanco seguidos, se tomaran como un único espacio, a excepción del comienzo de la línea, donde el espacio en blanco es ignorado.

Para ingresar espacios irrompibles en HTML se utiliza el código ** ** que significa *non breacking space*. Además, también existen otros códigos similares para introducir caracteres internacionales, como los acentuados. Por ejemplo para escribir el carácter "á", se debe utilizar el código **á** el carácter "é" se escribe como é y la "ó" como **&0acute;** Otro carácter que puede traer problemas es la "ñ": **ñ.**

4.4 Dar Formato al Texto

HTML ofrece un conjunto de etiquetas bastante amplio para dar formato al texto.

Una de las más importantes es la etiqueta **<P>**, de párrafo. La misma tiene etiqueta de cierre </**P**>, aunque es opcional. También tiene la propiedad de **align**, mediante la cual se puede especificar la alineación del párrafo: **left:** izquierda, **center:** centro, **right:** derecha o **justify:** justificar: la alineación por defecto es a la izquierda. He aquí un ejemplo de su uso.

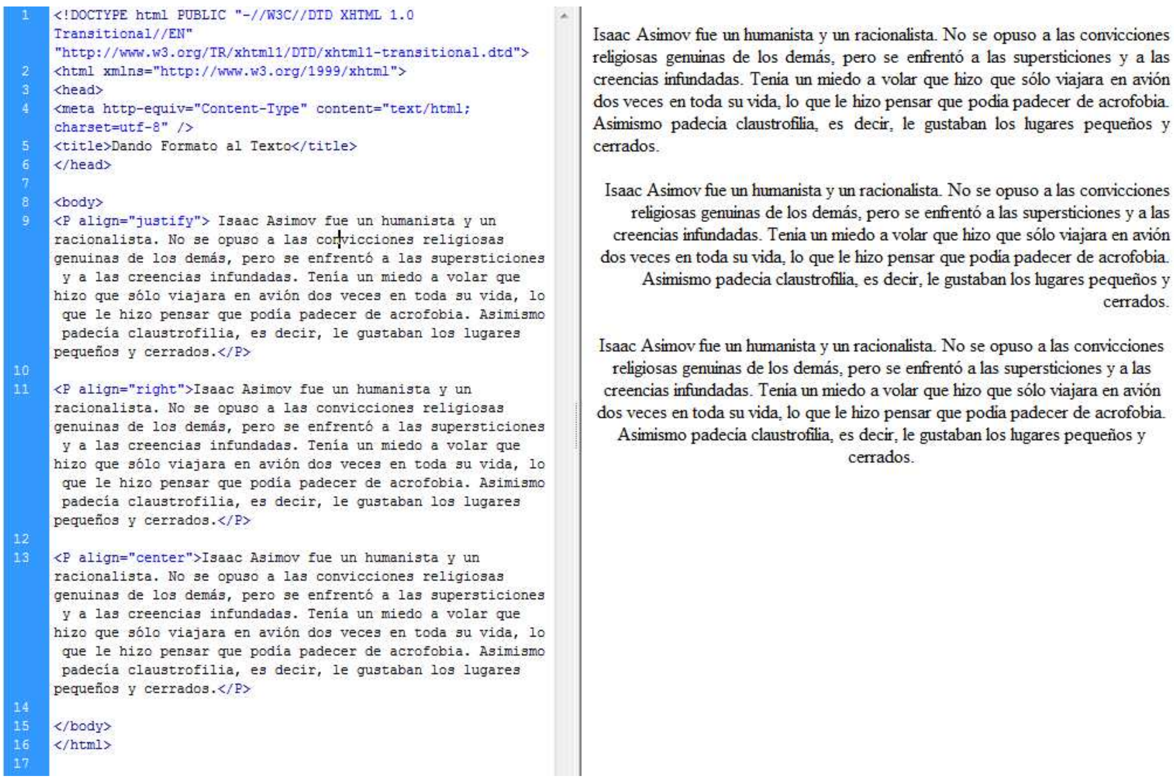


Imagen 4.2 Dando Formato al Texto.

Las etiquetas <**P align=”center”**> y </**P**> pueden reemplazarse por <**CENTER**> y </**CENTER**>**,** pero no existe un equivalente para reemplazar otro tipo de alineación, es decir no se puede reemplazar <**P align=”right”**> y </**P**> con <**RIGHT**> **y** </**RIGHT**> ya que no existen.

Otras etiquetas muy utilizadas son las que no están directamente relacionadas con el tratamiento de texto, son <**HR**> **(Horizontal Rule)** y <**BR**> **(Break Line).** La primera inserta una línea horizontal y la segunda un salto de línea, dentro del mismo párrafo.

Se observa que <**BR**> no tiene una etiqueta de cierra. Por otro lado, <**HR**> atributos de tamaño (**size**), alineación (**center**) y ancho (**width**). El ancho se puede expresar en porcentaje y se toma como referencia (100%) el ancho de la ventana. Por lo tanto, si se cambia el ancho de la ventana, el ancho del elemento variará. Si no se coloca el símbolo de porcentaje, se entiende que el ancho esta expresado en pixeles.

La etiqueta <**DIV**> también permite definir párrafos, aunque provoca una distancia más pequeña entre los mismos. Esta etiqueta se ha comenzado a utilizar mucho para definir capas, por lo que ofrece muchas más posibilidades que <**P**>.

También tenemos la etiqueta <**SPAN**>, la cual tiene propiedades semejantes a <**DIV**>, con la diferencia de que no inserta un cambio de línea.

Como ya dijimos, la etiqueta <**CENTER**> sirve para centrar el contenido que se encuentre en el medio. Sin embargo, es muy común reemplazarla por la etiqueta <**DIV align=”center”> y </DIV>.**

4.5 Otras etiquetas de formato.

Si deseamos dar al texto un color, tamaño y una fuente tipográfica determinados, podemos hacer uso de las etiquetas <**FONT**> y <**BASEFONT**>. La primera permite dar formato a cualquier texto, mientras que la segunda se suele colocar por única vez luego de <**BODY**>, y establezca las características por defecto del texto de la página. Ambas tienen tres atributos:

1. size: indica el tamaño, que puede variar entre 1 y 7 o valores relativos al establecido antes de la etiqueta, como +1, -2, etc.

2. color: indica el color del texto y puede ser un número hexadecimal o el nombre en inglés. El valor hexadecimal de los colores siempre debe ir precedido por el carácter #.

3. face: define el nombre de la fuente.

Es importante recordar que no es posible colocar cualquier fuente en el atributo face, ya que quien visita la pagina debe tenerla instalada en su sistema para poder visualizarla la fuente por defecto (Times New Roman o Helvetica, en la mayoría de los casos). Sin embargo, es posible definir grupos de fuentes, separados por la coma, que irán presentando diversas alternativas en caso de que la fuente principal no esté instalada en el sistema del usuario.

Otras etiquetas muy útiles en HTML son las de los estilos negrita, itálica y subrayado.

La forma de aplicarlas es la siguiente:

Negrita: <B> Texto </B>

Cursiva: <I> Texto </I>

Subrayada:< U> Texto </U>

También disponemos de un conjunto de 6 etiquetas de cabeceras para el ingreso de títulos de distinta jerarquía. Las mismas van desde <**H1**> hasta <**H6**>.

Otras etiquetas útiles son las que permiten aumentar o disminuir un poco el tamaño del texto que se viene usando. Estas son <**BIG**>, la cual agranda el texto, y <**Small**>, con la cual se achica. Obviamente, es necesario colocar la etiqueta de cierre.

La etiqueta <**TT**> da al texto el formato teletipo, en donde todos los caracteres ocupan el mismo espacio. La etiqueta <**CODE**>, para el ingreso para el ingreso de código de programación, da al texto un formato de programación, da al texto un formato similar.

Otra etiqueta que podría sernos de utilidad es <**PRE**>. Esta tiene la particularidad de que el texto que se escriba entre la apertura y el cierre se mostrara respetando los espacios y los saltos de línea tal cual lo escribíamos en el archivo de texto.

Finalmente, tenemos las etiquetas <**SUB**> y <**SUP**> para ingresar subíndices y superíndices respectivamente.

4.6 Formularios.

Los formularios son de mucha utilidad cuando se trabaja con páginas dinámicas en lenguaje PHP.

Un formulario **HTML** por sí solo no puedo hacer mucho. Simplemente se limita a enviar los valores al servidor pero, si allí no hay nada que los procese, será inútil. Para definir un formulario, primero hay que crearlo utilizando las etiquetas <**FORM**> y </**FORM**> y luego incluir los distintos elementos que lo componen entre medio. Estos elementos pueden ser campos de texto, listas de selección, botones de opción, casillas de verificación, botones, etc.

4.7 Definir los límites del formulario.

La etiqueta <**FORM**> marca el inicio del formulario HTML. La misma necesita algunos atributos para saber cómo manejar los datos del formulario y qué hacer con ellos.

La propiedad **action** define a que pagina se enviaran los datos para procesarlos o que se hará con los datos una vez que se pulse sobre el botón Enviar. También tenemos **method** que define la forma en que se envían los resultados; aquí tenemos dos posibilidades:

Get: los datos del formulario son enviados por la URL del navegador.

Post: Los datos son enviados a través de las cabeceras http y son invisibles para el usuario.

También tenemos otras propiedades como **name**, con la cual se puede definir un nombre para el formulario (muy útil para aplicaciones con JavaScript); **target**, similar al atributo de los hipervínculos; **style** y otras más.

4.8 Campos de Texto

Los campos de texto es un elemento que utiliza con mayor frecuencia. Los mismos permiten que el usuario ingrese los datos de una línea como su nombre, dirección de correo electrónico, teléfono, etc.

La forma de insertar este tipo de de elemento es utilizando la etiqueta <**INPUT>**, la cual no tiene etiqueta de cierre, con el atributo **type=”text”.** Además, tenemos otra propiedad muy importante y que no puede faltar: **name**.

El atributo **value** nos permite definir el valor por defecto del elemento.

El mismo puede ser luego borrado por el usuario para anotar sus propios datos.

También se tiene el atributo **size**, para especificar el ancho del campo en caracteres, y **maxlength**, con el cual se puede establecer un límite para que una vez alcanzado no se pueda ingresar más texto.

4.9 Casillas de Verificación

Estos elementos permiten que el usuario seleccione distintas opciones, pueden ser una o más. Se le suele utilizar para la elección de productos a comprar, listas de correo a suscribirse, etc.

También se ingresan con la etiqueta <**INPUT>**, salvo que en este caso variara el tipo (type) de elemento.

El atributo **name** sirve para identificar a la opción y **value** para incluir el contenido que se considerara, en caso de que la persona marque estas opciones.

Para las casillas de verificación también tenemos la propiedad **checked**, la cual se inserta sin valor, que marca una casilla por defecto.

4.10 Botones de Opción

Este elemento también es conocido como un botón de radio. Es en cierto aspecto similar al anterior, pero en este caso, los elementos se agrupan, utilizando el mismo nombre, y solo está permitida una única selección. Solo es posible seleccionar dos botones de opción si estos tienen nombres diferentes.

Para definir un botón de opción se utiliza nuevamente la etiqueta **INPUT** y se debe especificar **type=”radio”.**

4.11 Áreas de Texto.

Si deseamos que el usuario ingrese un texto largo, deberíamos ofrecerle un espacio más cómodo para escribir. Es común ver áreas de texto en los campos para redactar mensajes de las páginas web que ofrecen un correo gratuito, como Yahoo! o Hotmail, por ejemplo.

Las áreas de texto se definen con las etiquetas <**TEXTAREA** > y </**TEXTAREA>** y el texto que va en el medio es el texto que aparecerá en el campo por defecto. También se puede definir su alto y ancho con los atributos **rows (filas)** y **cols (columnas)** respectivamente. También debe llevar un atributo **name,** como los demás elementos de formulario.

Este elemento suele ser muy engañoso a la hora de determinar su tamaño. Es muy difícil lograr que en todos los navegadores se vea igual. Por lo tanto, cuando definamos una medida para el mismo, conviene probarlo en distintos programas, para observar como se ve y fijarse que no afecte el diseño de nuestra página.

4.12 Todo Tipo de Botones y Comandos.

Existen dos tipos de comandos muy utilizados en los formularios; estos son los **submit** y **reset**. El primero envía su formulario para su procesamiento a donde hayamos indicado en la propiedad **action** de la etiqueta <**FORM>**, y el segundo limpia todo el texto y opción que hayamos ingresado, dejando el formulario tal y como estaba al comienzo.

Estos dos elementos también se ingresan con <**INPUT>** de la siguiente manera:

```
<INPUT type=”submit” name=”cmdEnviar” value=” Enviar Formulario” >
```

```
<INPUT type=”submit” name=”cmdEnviar” value=” Enviar Formulario” >
```

En este caso, el valor de la propiedad **value** determinara el texto que aparecerá sobre el botón. Los atributos **name** no son necesarios, sobretodo en el botón **cmdLimpiar**.

También podemos crear botones genéricos para realizar algún tipo de acción predeterminada, la cual deberemos programar, con JavaScript o algún otro lenguaje:

```
<INPUT type=”button” value=” Etiqueta” onClick= “action()” >
```

Para los botones también tenemos las etiquetas <**BUTTON>** Y <**/BUTTON>**. Son equivalentes a las anteriores, y la forma de definir los tipos de botones antes vistos es la siguiente:

```
<BUTTON type=”submit”> Enviar Formulario </BUTTON>

<BUTTON type=”reset”> Limpiar Formulario </BUTTON>

<BUTTON type=”button”> Texto del Boton </BUTTON>
```

Por último, también es posible crear un botón de **submit** a partir de una imagen, una práctica muy habitual para dar a los botones un aspecto más vivo.

4.13 Menús de Selección

Un menú desplegable contiene diferentes opciones entre las cuales elegir. Básicamente su función es muy similar a la de los botones de opción o las casillas de verificación, con la diferencia de que en el caso de los menús es posible ocupar poco espacio y colocar un mayor número de opciones. Es posible crear menús de selección simple o múltiple. Los de selección múltiple permiten elegir varias opciones a la vez usando la tecla **Ctrl**.

Para crear un menú se utilizan dos etiquetas diferentes <**SELECT>** y <**OPTION>:**

Observen que el atributo **name** va en la etiqueta <**SELECT>** y **value** va en cada etiqueta <**OPTION>**. El texto que va entre las etiquetas de captura y cierre de <**OPTION>** es el que aparecerá en el menú, y el de **value** será el que recibirá el programa o pagina que procesara los datos. No es necesario que estos valores sean iguales, de hecho, en algunos casos puede ser útil un número como valor, para identificar el país, por ejemplo.

La etiqueta <**SELECT>** también tiene otros atributos, como **size**, que define el alto en líneas y la propiedad múltiple, la cual no lleva valor y define la posibilidad de seleccionar más de una opción a la vez. Por su lado, <**OPTION>** puede tener la propiedad **selected** en algunos de sus elementos (solo uno si no está múltiple en **<SELECT>**) para que aparezca seleccionado en forma predeterminada.

4.14 Otros Elementos

Finalmente nos quedan dos elementos que no son muy utilizados, uno de ellos es < **LABEL>**, el cual permite crear una etiqueta para un campo de un formulario. La única ventaja que tiene es para el caso de los botones de radio y las casillas de verificación, ya que si creamos una etiqueta para estos elementos, al cliquear sobre ella, se marcara la opción directamente. Para declarar una etiqueta para un elemento de formulario determinado hay que utilizar las propiedades **id**, en el elemento, y **for**, en la etiqueta.

Otro ejemplo novedoso es el set de campos para formulario. Reúne en un marco un grupo determinado de campos. Simplemente hay que ubicar los campos que queremos agrupar en las etiquetas <**FIELDSET** > y **</FIELDSET>**.

4.15 Tablas

El uso de las tablas en las páginas web se encuentra muy extendido. Las mismas no solo guardan su uso original, para la presentación ordenada de datos, sino que también aquí se utilizan para alineación y presentación del contenido. Son muy pocas las etiquetas necesarias para el trabajo con tablas: <**TABLE>** y **</TABLE>**, para definir el comienzo y fin de la tabla; <**TR** >y **</TR>** para definir el comienzo de una nueva fila.

Hoy en día, Muchas de las funcionalidades de las tablas **HTML**, sobre todo aquellas relacionadas con la alineación y organización de elementos, está siendo reemplazada por las capas.

Unidad 5 MySQL.

5.1 Definición de Base de Datos.

Una base de datos es una colección de archivos interrelacionados, son creados con un DBMS (Sistema Manejador de Bases de Datos). El contenido de una base de datos engloba a la información concerniente (almacenadas en archivos) de una organización (por ejemplo), de tal manera que los datos estén disponibles para los usuarios, una finalidad de la base de datos es eliminar la redundancia o al menos minimizarla. Los tres componentes principales de un sistema de base de datos son el hardware, el software DBMS y los datos a manejar, así como el personal encargado del manejo del sistema.

5.2 Entidad Relación.

El modelo Entidad-Relación se basa en una percepción del mundo real, la cual está formada por objetos básicos llamados entidades y las relaciones entre estos objetos así como las características de estos objetos llamados atributos.

Una ***entidad*** es un objeto que existe y se distingue de otros objetos de acuerdo a sus características llamadas atributos. Las entidades pueden ser concretas como una persona o abstractas como una fecha.

Un ***conjunto de entidades*** es un grupo de entidades del mismo tipo. Por ejemplo el conjunto de entidades CUENTA, podría representar al conjunto de cuentas de un banco X, o ALUMNO representa a un conjunto de entidades de todos los alumnos que existen en una institución. Una entidad se caracteriza y distingue de otra por los ***atributos***, en ocasiones llamadas propiedades, que representan las características de una entidad. Los atributos de una entidad pueden tomar un conjunto de valores permitidos al que se le conoce como **dominio** del atributo. Así cada entidad se describe por medio de un conjunto de parejas formadas por el atributo y el valor de dato. Habrá una pareja para cada atributo del conjunto de entidades.

Nombre_atributo, Valor

No_control , 96310418

Nombre , Sánchez Osuna Ana

Esp , LI

5.3 Tipos de datos.

Para cada campo de una tabla es necesario determinar un tipo de dato que contiene o va a contener, para de esta forma ajustar el diseño de la base de datos y conseguir un almacenamiento óptimo con la menor utilización de espacio posible. Los tipos de datos con los que podemos contar en MySQL son:

Tipos numéricos:

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

TinyInt: es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255.

Bit ó Bool: un número entero que puede ser 0 ó 1.

SmallInt: número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

Integer, Int: número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295.

BigInt: número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float: número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

xReal, Double: número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308.

Decimal, Dec, Numeric: Número en coma flotante desempaquetado. El número se almacena como una cadena.

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

Tabla 5.3.1 Tipos de datos numéricos.}

Tipos fecha:

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes está comprendido entre 0 y 12 y que el día está comprendido entre 0 y 31.
Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

DateTime: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

TimeStamp: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Tamaño	Formato
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammddhhmmss
8	ñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

Tabla 5.3.2 Tamaño tipos de dato fecha.

Time: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

Year: almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Tabla 5.3.3 Tipos de dato fecha.

Tipos de cadena:

Char(n): almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n): almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)

La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

TinyText y TinyBlob: Columna con una longitud máxima de 255 caracteres.

Blob y Text: un texto con un máximo de 65535 caracteres.

MediumBlob y MediumText: un texto con un máximo de 16.777.215 caracteres.

LongBlob y LongText: un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

Enum: campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

Set: un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

Tabla 5.3.4 Tipos de dato cadena.

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
''	''	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Tabla 5.3.5 Diferencia de almacenamiento entre los tipos Char y VarChar.

5.4 Crear una Base de Datos.

Para crear una base de datos desde nuestro manejador EMS MySQL previamente abierto, es necesario dirigirnos a la opción Create o Create Data Base, en la imagen 5.4.1 estas opciones se encuentran resaltadas en color amarillo.

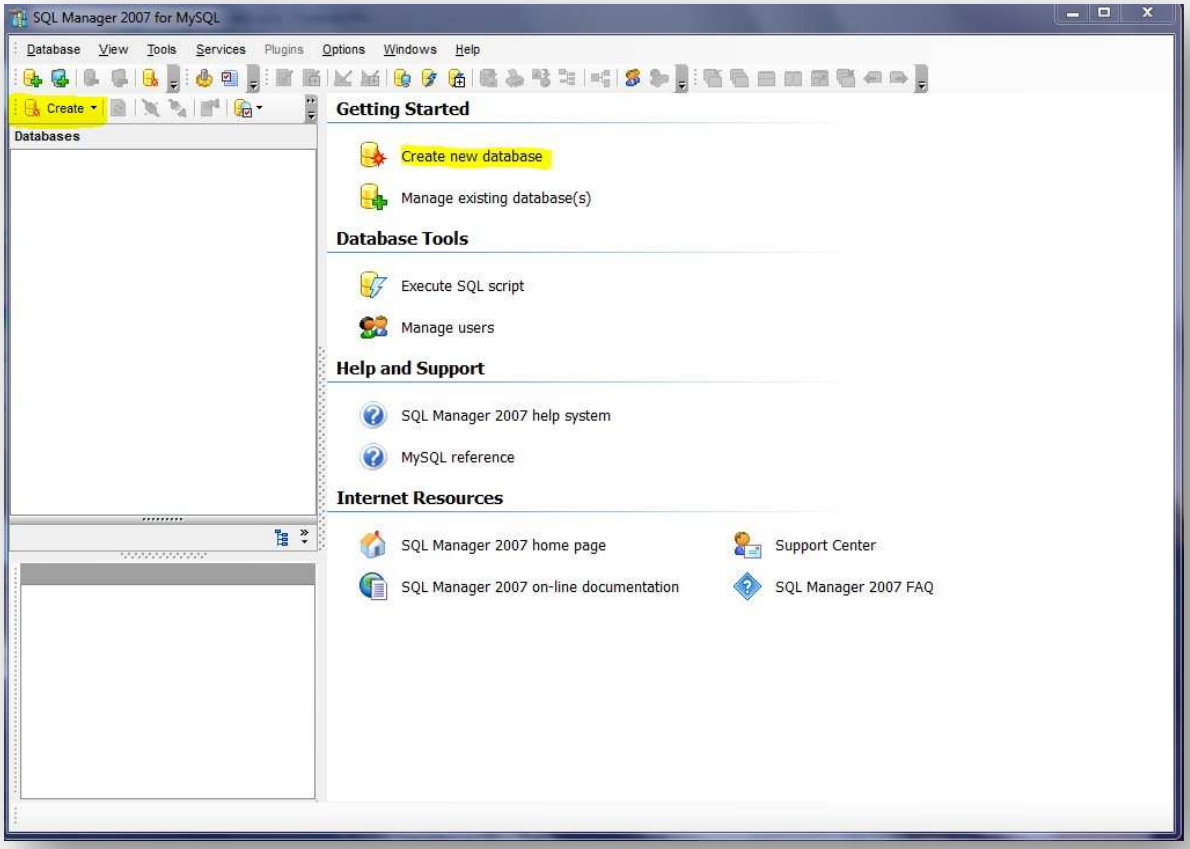


Imagen 5.4.1 Creando una base de datos.

Damos click en esa opción la cual nos mostrara el siguiente menú:

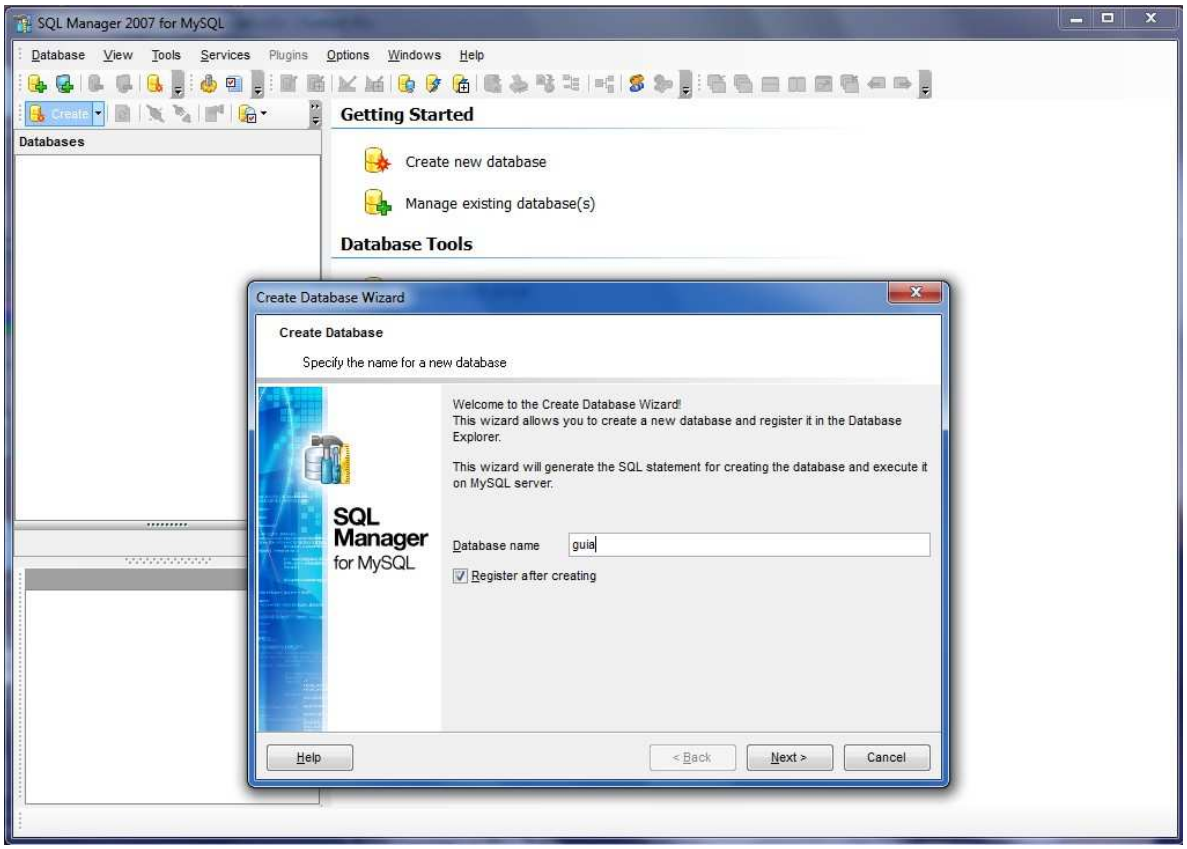


Imagen 5.4.2 Menú de creación de una Base de Datos.

En este menú tenemos la opción de poner el nombre que deseamos para nuestra Base de Datos. Y damos click en Next. Dejaremos habilitada la opción de “Register After Creating” esto con la finalidad de que al terminar de crear la base de datos la podamos registrar en el manejador y de esta manera podamos manipularla mas fácilmente.

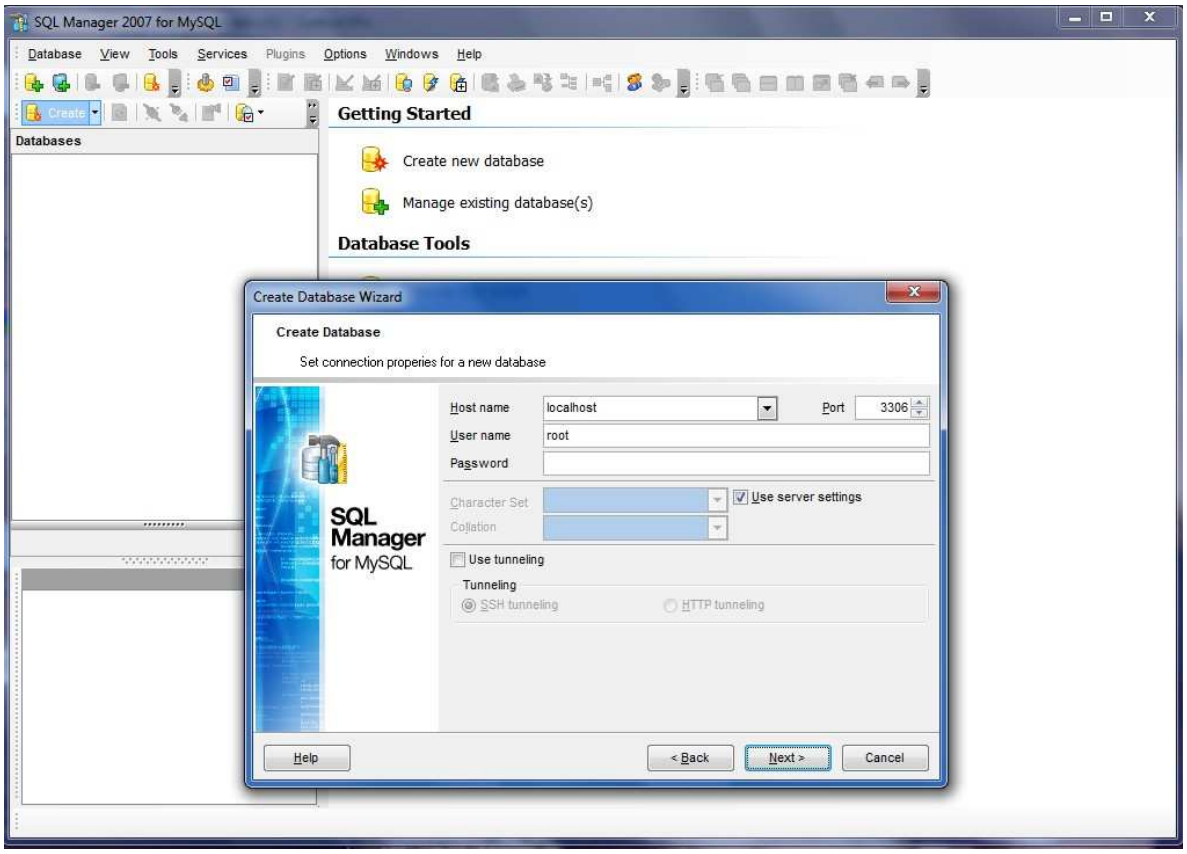


Imagen 5.4.3 Propiedades de la Base de Datos a crear.

Como se muestra en la imagen podemos configurar los diferentes parámetros de nuestra base de datos que estamos a punto de crear, los parámetros como el nombre del host y el nombre de usuario que se muestran son los que aparecen por default por el manejador, en el caso de que estos parámetros se hayan puesto de una manera distinta al momento de la instalación de nuestro servidor tendríamos que especificar cuáles son. Para continuar daremos click en next.

Al dar click en next nos mostrara la sintaxis SQLÑ que el manejador va a ejecutar para la creación de la base de datos.

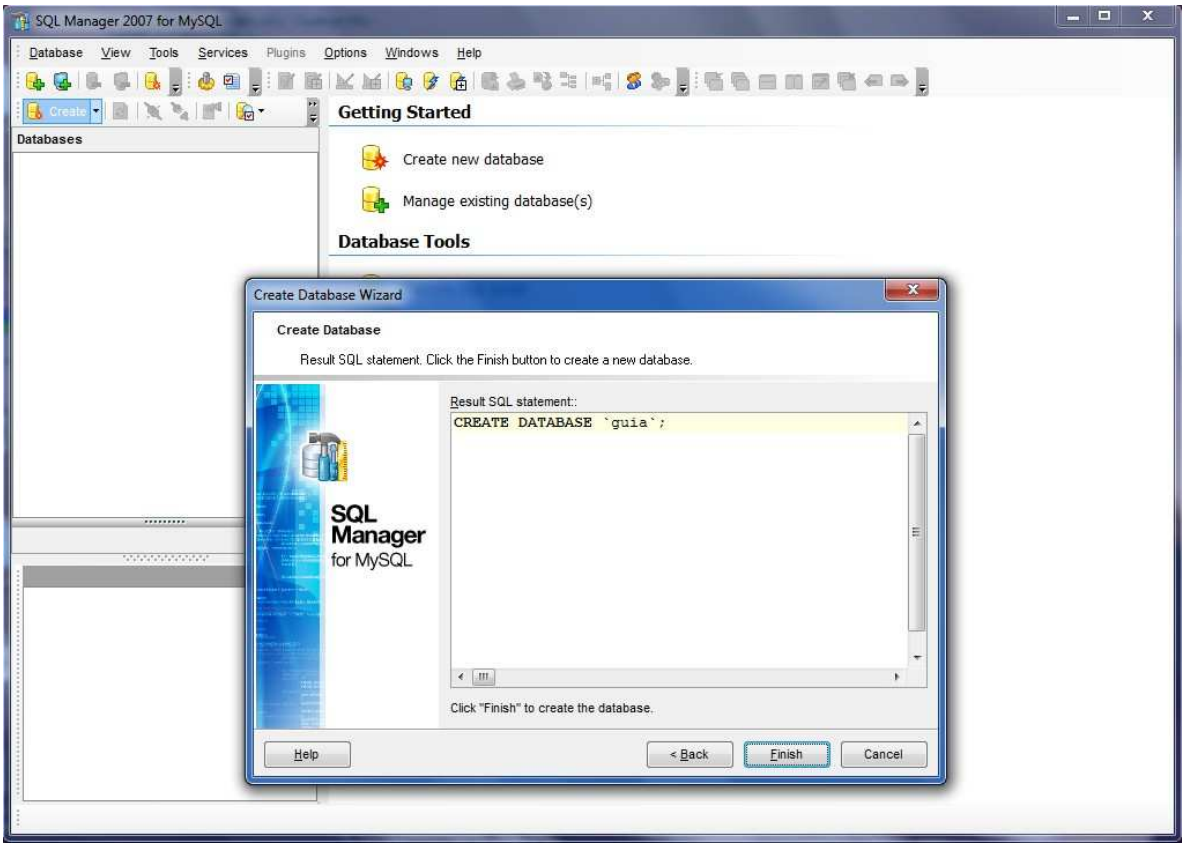


Imagen 5.4.4 Sentencia SQL a ejecutar.

Al mostrarnos esta pantalla daremos click en Finish para terminar con la creación de nuestra Base de Datos.

Como se comentaba en un principio dejamos habilitada la opción de registrar nuestra base de datos después de su creación, así que nos mostrara una pantalla como la de la siguiente imagen.

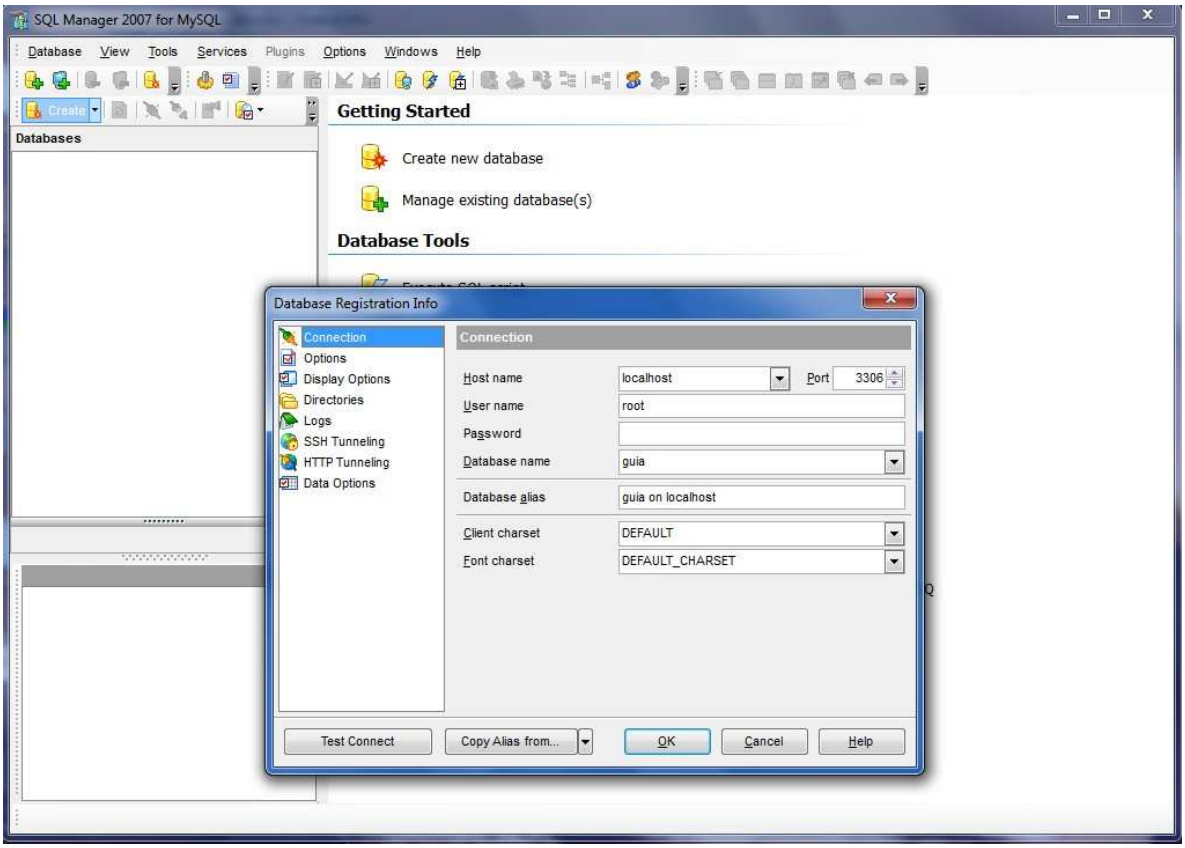


Imagen 5.4.5 Registro de la base de datos en el manejador.

Al mostrarnos esta pantalla lo único que tendremos que hacer es dar click en OK de esta forma nuestro manejador quedara listo para manipular la base de datos que hemos creado.

5.5 Creación de una Tabla.

Ahora crearemos una tabla desde nuestro manejador. Para esto es necesario irnos a la opción de **TABLES** como se muestra en la imagen y daremos un click para continuar.

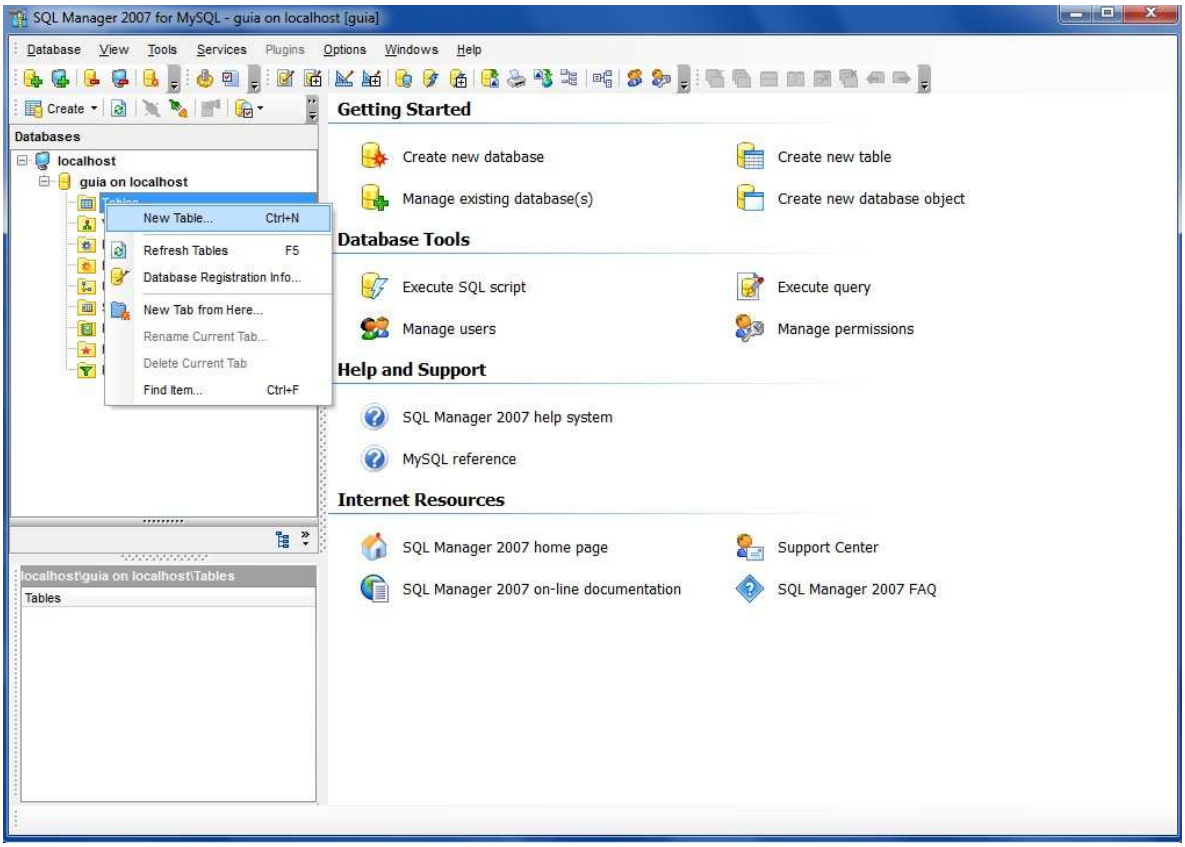


Imagen 5.5.1 Creación de una tabla.

Al dar click en la opción “New Table” nos desplegara un nuevo menú donde podremos configurar los diferentes parámetros de nuestra tabla como el nombre. En nuestro ejemplo de la imagen 5.5.2 le asignamos el nombre de ejemplo1. Para que la tabla sea creada es necesario que demos un click en la opción compile, con esto nuestra tabla será guardada con el nombre que elegimos

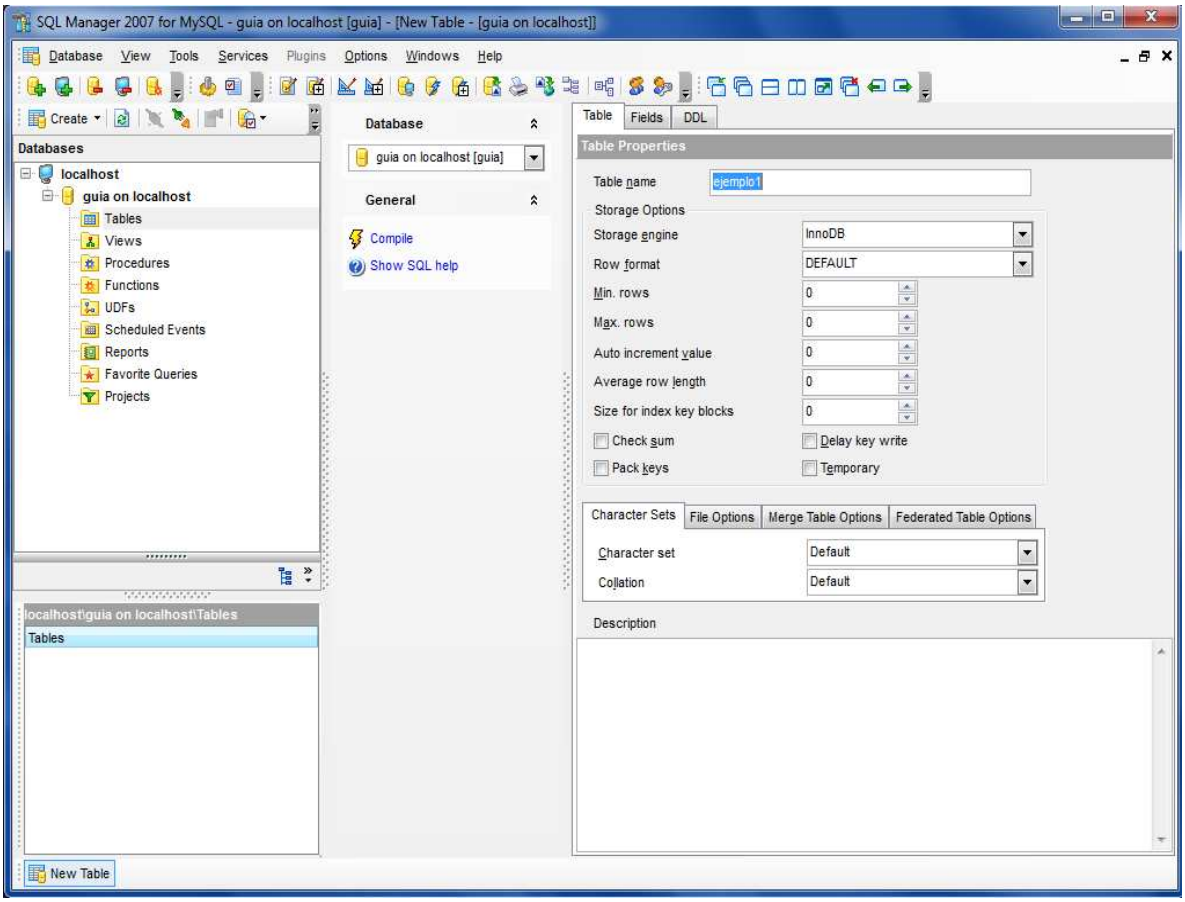


Imagen 5.5.2 Configuración de parámetros en la creación de una tabla.

5.6 Creando, Modificando y Eliminando un Registro.

Para la creación de un registro nos vamos a la pestaña llamada **fields** o usar la opción "**add new field"** la cual nos mostrara una pantalla como la imagen 5.6.1.

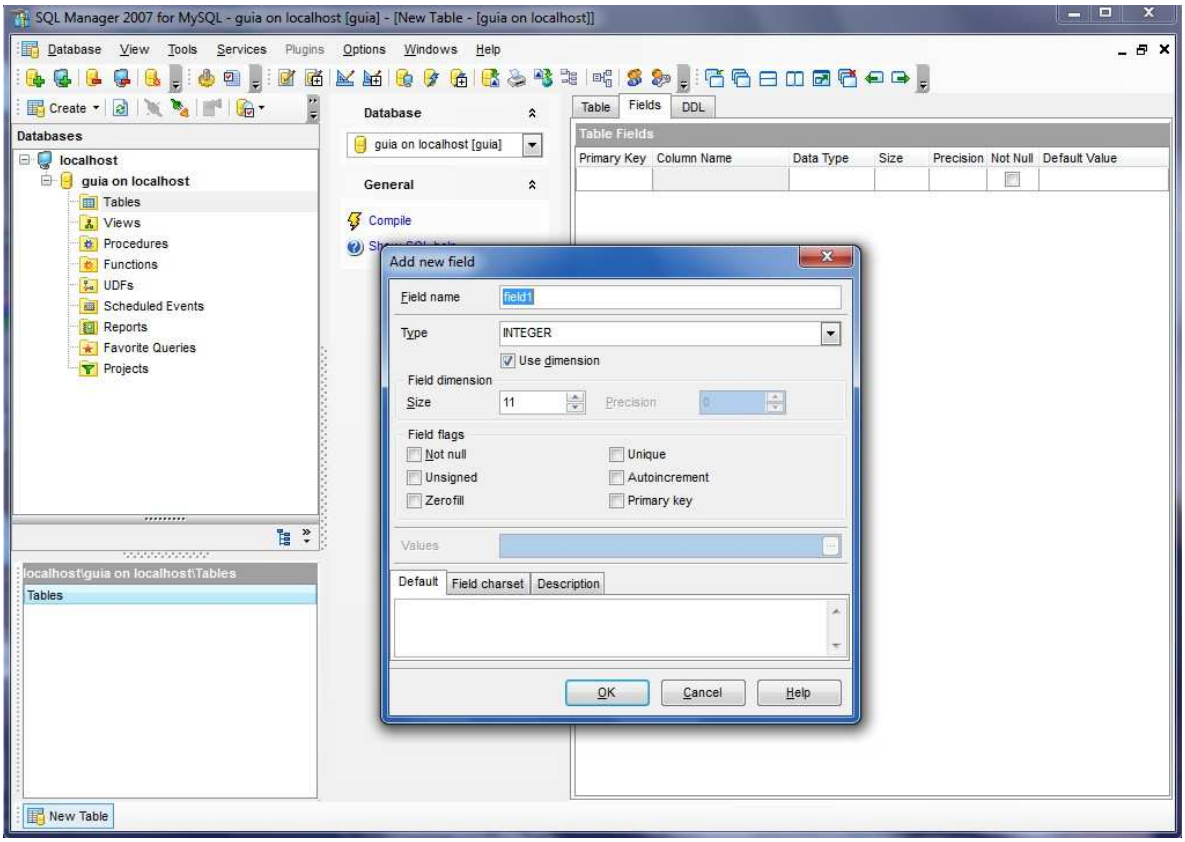


Imagen 5.6.1 Creando registros.

En esta opción podremos configurar parámetros tales como el nombre en la casilla **field name**, el tipo de dato en la casilla **type** y el tamaño del campo en la casilla **size**. En la parte inferior aparecen otras propiedades como llave primaria, not null, etc.

Una vez configurados estos parámetros damos click en OK y el campo habrá sido creado. De ser necesario bastara con dar un clic en compile para proceder con la creación del campo.

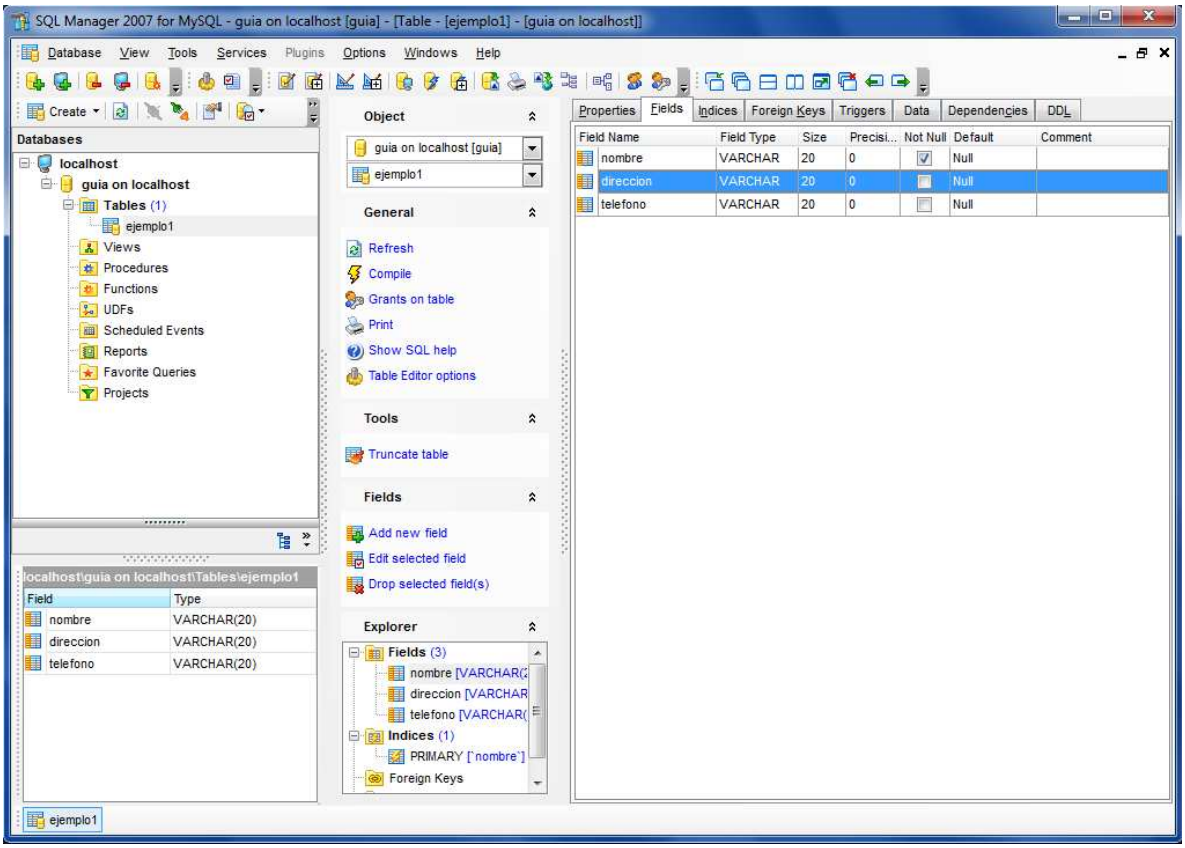


Imagen 5.6.2 Vista final de los campos agregados.

Para modificar algún parámetro de algún campo, lo podremos hacer simplemente con hacer doble click sobre el campo a modificar y de esta forma nos desplegara el menú donde podremos editar todas estas opciones.

Para eliminar un campo lo que tendremos que hacer es simplemente seleccionar el campo a eliminar y seleccionar la opción **"Drop selected field(s)"** la cual nos mostrara una advertencia para comprobar si en realidad queremos borrar ese campo. De ser hay que dar click en OK para confirmar la eliminación del campo.

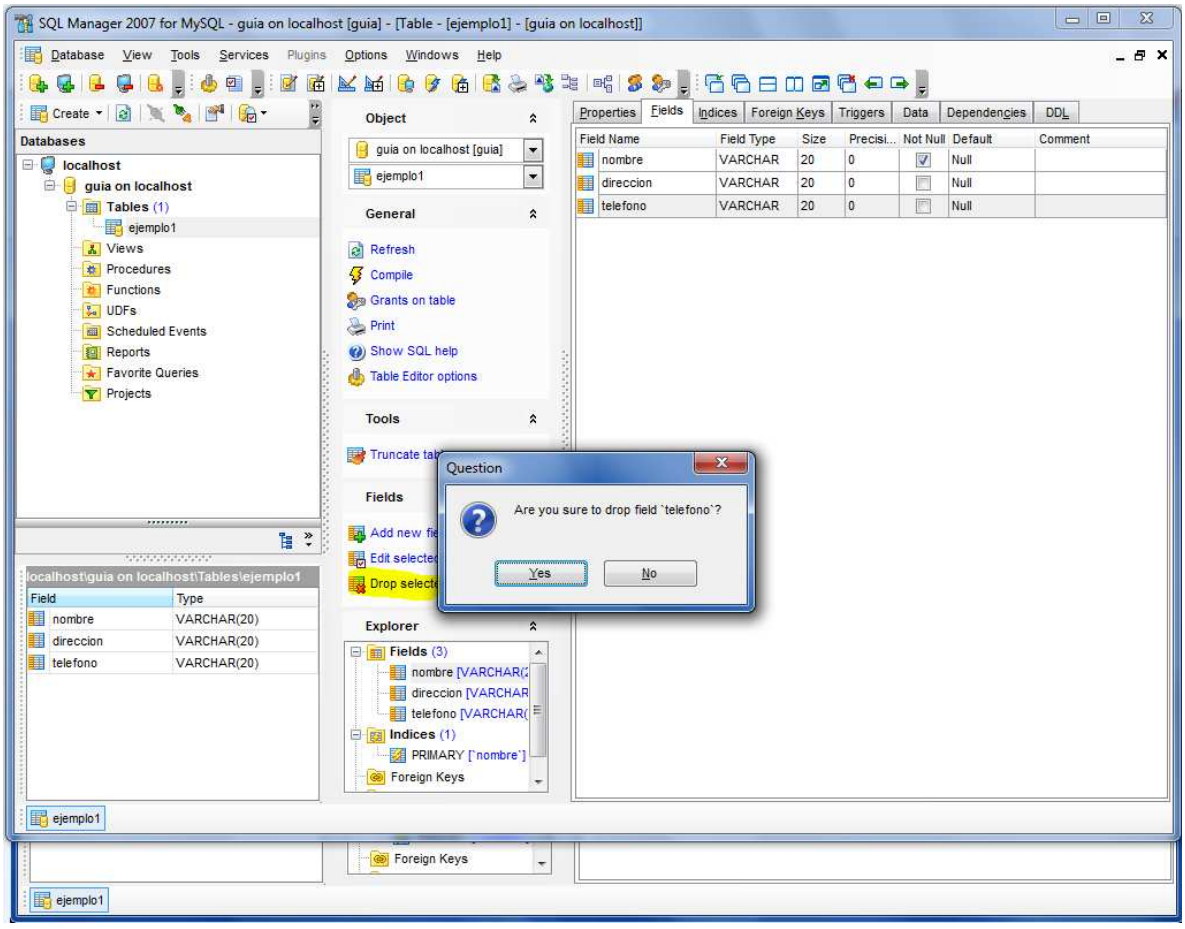


Imagen 5.6.3 Pantalla de eliminación de un campo. La opción para eliminar esta resaltada en amarillo.

5.7 Consultar Datos.

Para la consulta de datos tendremos que irnos a la opción "**Show Query Builder**" desde la cual podremos hacer nuestras consultas de manera visual o desde código SQL.

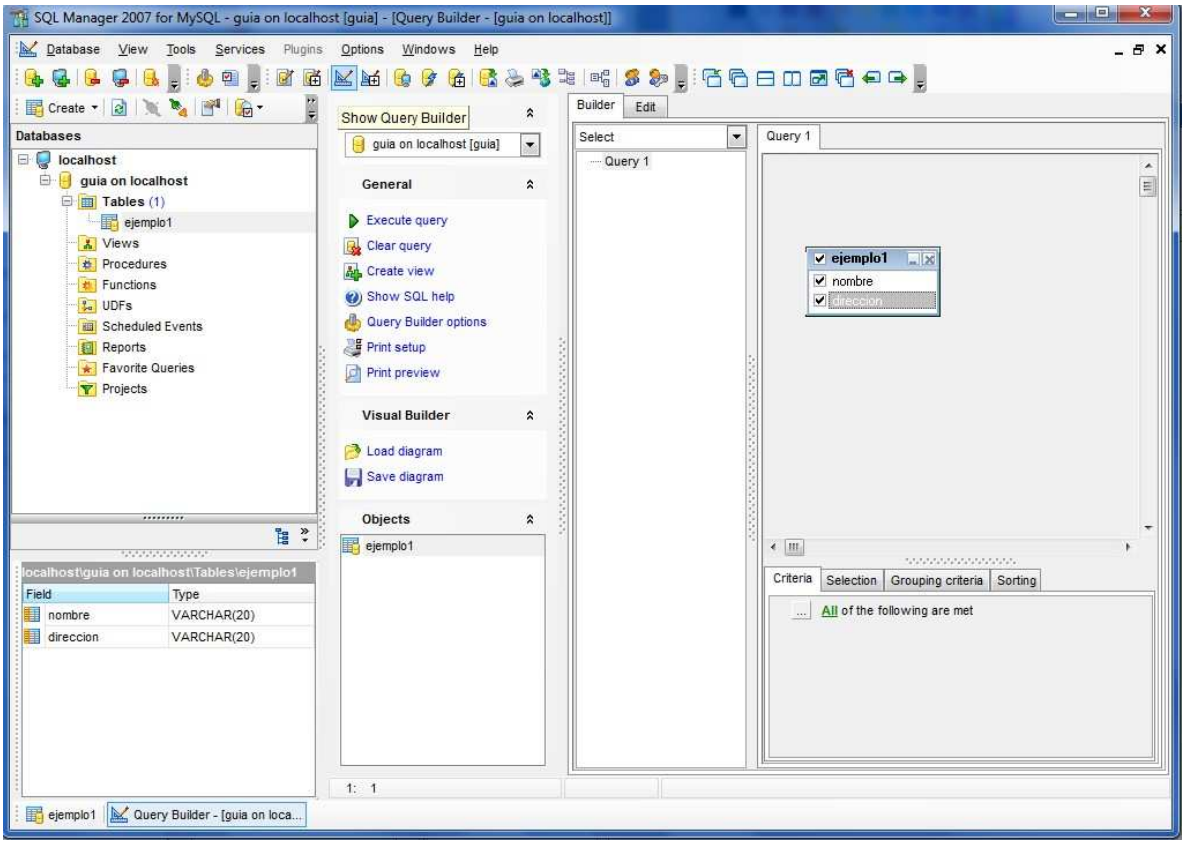


Imagen 5.7.1 Pantalla del query builder.

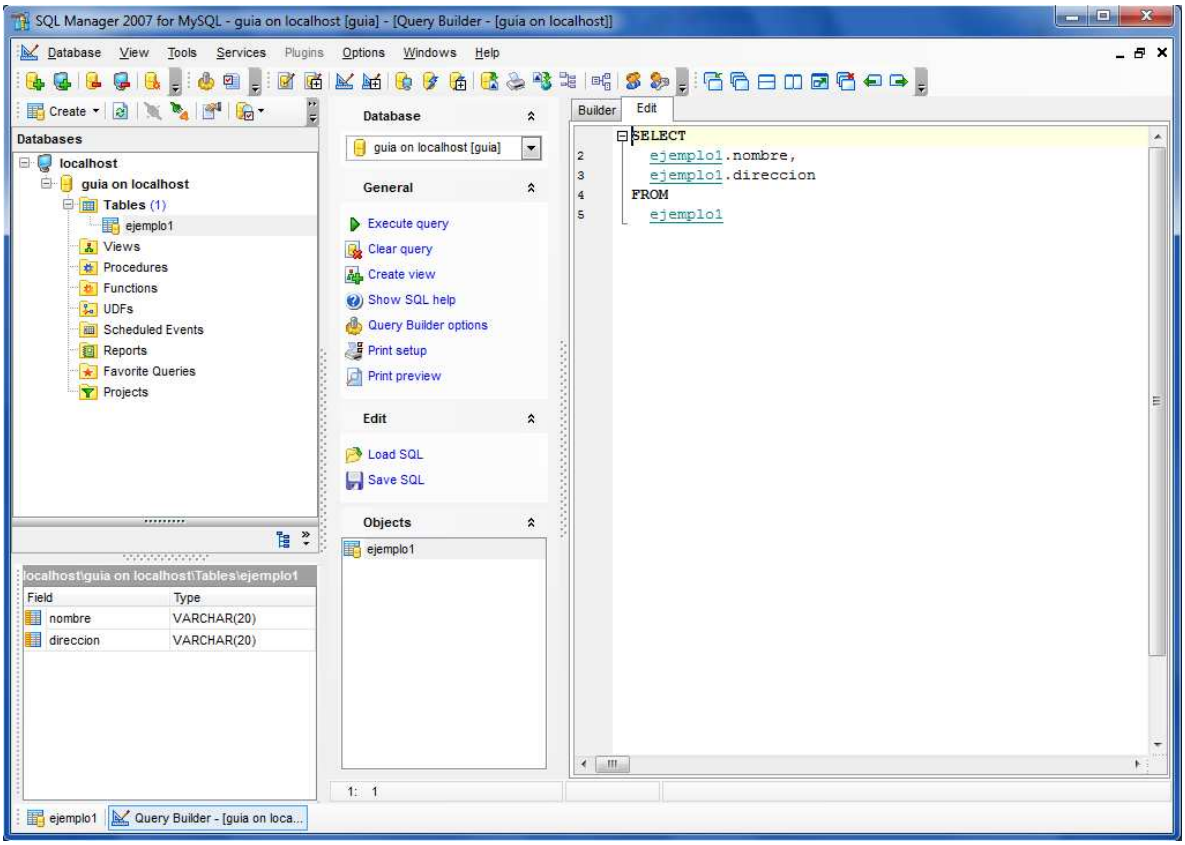
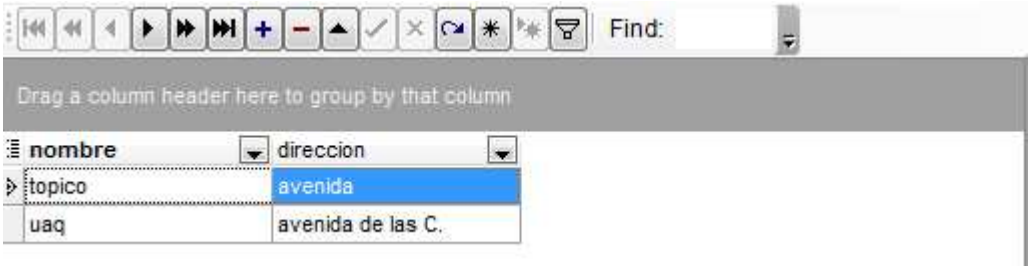


Imagen 5.7.2 Pantalla del query builder que muestra la sentencia SQL a ejecutar.

Como se muestra en la imagen 5.7.2 el manejador permite que veamos la sentencia sql a ejecutar y nos da la opción de editar ahí mismo, al terminar la sentencia hay q dar click en la opción "**execute query**" Con esto nos desplegara la información solicitad o nos mandara un mensaje de error indicándonos en que línea de código tenemos problemas.



6.-PHP.

6.1 Delimitadores.

Hay cuatro formas de salir de HTML y entrar en el "modo de código PHP":

Ejemplo 1.

```
<?
echo ("esta es la más simple, una instrucción de procesado
SGML\n");
?>
```

Ejemplo 2.

```
<?php
echo("si quiere servir documentos XML, haga esto\n");
?>
```

Ejemplo 3.

```
<script language="php">
echo ("a algunos editores (como FrontPage) no lesgustan las
intrucciones de procesado");
</script>
```

Ejemplo 4.

```
<% echo ("Puedes también usar etiquetas tipo ASP"); %>
<%= $variable; # Esto es una forma abreviada de "<%echo .."
%>
```

La primera forma sólo está disponible si se han habilitado las etiquetas cortas. Esto se puede hacer a través de la función ***short_tags()***, habilitando la opción de configuración ***short_open_tag*** en el archivo de configuración de PHP, o compilando PHP con la opción ***enable-short-tags*** en configuracion.

La cuarta manera está disponible sólo si se han habilitado las etiquetas tipo ASP usando la opción de configuración ***asp_tags***.

Nota: El soporte para las etiquetas tipo ASP se añadió en 3.0.4.

La etiqueta de cierre de un bloque incluirá el carácter de nueva línea final si hay uno presente.

Separación de instrucciones

Las instrucciones se separan igual que en C o perl - terminando cada sentencia con un punto y coma.

La etiqueta de cierre (?>) también implica el fin de la sentencia, así lo siguiente es equivalente:

Ejemplo 5.

```
<?php
echo "Esto es una prueba";
?>
```

Ejemplo 6.

```
<?php echo "Esto es una prueba" ?>
```

6.2 Comentarios.

PHP soporta comentarios tipo 'C', 'C++' y shell de Unix. Por ejemplo:

Ejemplo 1.

```
<?php
echo "Esto es una prueba"; // Esto es un comentario tipo c++
para una línea
/* Esto es un comentario multilínea
otra línea más de comentario*/
echo "Esto es aún otra prueba";
echo "Una Prueba Final"; # Este es un comentario tipo shell
?>
```

El tipo de comentario de "una línea" sólo comenta, en realidad, hasta el fin de la línea o el bloque actual de código PHP, lo que venga primero.

Ejemplo 2.

```
<h1>Esto es un <?# echo "simple";?> ejemplo.</h1>
<p>La cabecera de arriba dirá 'Esto es un ejemplo'.
```

Se debería tener cuidado para no anidar comentarios de tipo 'C', lo cual puede ocurrir cuando se comentan grandes bloques.

Ejemplo 3.

```
<?php
/*
echo "Esto es una prueba"; /* Este comentario causará un
problema */
```

```
*/
?>
```

6.3 Variables.

6.3.1 Conceptos Básicos

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Ejemplo 1.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce la salida "Bob, Joe"
```

En PHP3, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor íntegro de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después e asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra. Para más información sobre este tipo de asignación, vea Expresiones.

PHP4 ofrece otra forma de asignar valores a las variables: asignar por referencia. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" o "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes arrays u objetos.

Para asignar por referencia, simplemente se antepone un ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida 'Mi nombre es Bob' dos veces:

Ejemplo 2.

```
<?php
$foo = 'Bob'; // Asigna el valor 'Bob' a $foo
$bar = &$foo; // Referencia $foo vía $bar.
$bar = "Mi nombre es $bar"; // Modifica $bar...
echo $foo; // $foo también se modifica.
echo $bar;
?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

Ejemplo 3.

```
<?php
$foo = 25;
$bar = &$foo; // Esta es una asignación válida.
$bar = &(24 * 7); // Inválida; referencia una expresión sin
nombre.
function test() {
return 25;
}
$bar = &test(); // Inválida.
?>
```

Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. De todas formas, muchas de esas variables no pueden estar completamente documentadas ya que dependen de sobre qué servidor se esté ejecutando, la versión y configuración de dicho servidor, y otros factores.

Algunas de estas variables no estarán disponibles cuando se ejecute PHP desde la línea de comandos.
A pesar de estos factores, aquí tenemos una lista de variables predefinidas disponibles en una instalación por defecto de PHP 3 corriendo como modulo de un Apache (http://www.apache.org/) 1.3.6 con su configuración también por defecto.

Para una lista de variables predefinidas (y muchas más información útil), por favor, vea (y use) **phpinfo().**

Nota: Esta lista no es exhaustiva ni pretende serlo. Simplemente es una guía de qué tipo de variables predefinidas se puede esperar tener disponibles en un script.

Variables de Apache

Estas variables son creadas por el servidor web Apache (http://www.apache.org/). Si se está utilizando otro servidor web, no hay garantía de que proporcione las mismas variables; pueden faltar algunas, o proporcionar otras no listadas aquí. Dicho esto, también están presentes las variables de la especificación CGI 1.1 (http://hoohoo.ncsa.uiuc.edu/cgi/env.html), por lo que también se deben tener en cuenta. Tenga en cuenta que unas pocas, como mucho, de estas variables van a estar disponibles (o simplemente tener sentido) si se ejecuta PHP desde la línea de comandos.

GATEWAY_INTERFACE

Qué revisión de la especificación CGI está usando el servidor; por ejemplo 'CGI/1.1'.

SERVER_NAME
El nombre del equipo servidor en el que se está ejecutando el script. Si el script se está ejecutando en un servidor virtual, este será el valor definido para dicho servidor virtual.

SERVER_SOFTWARE
Una cadena de identificación del servidor, que aparece en las cabeceras al responderse a las peticiones
.
SERVER_PROTOCOL
Nombre y revisión del protocolo a través del que se solicitó la página; p.ej. 'HTTP/1.0';

REQUEST_METHOD
Qué método de petición se usó para acceder a la página; p.ej. 'GET', 'HEAD', 'POST', 'PUT'.

QUERY_STRING
La cadena de la petición, si la hubo, mediante la que se accedió a la página.

DOCUMENT_ROOT
El directorio raíz del documento bajo el que se ejecuta el script, tal y como está definido en el fichero de configuración del servidor.

HTTP_ACCEPT
Los contenidos de la cabecera Accept: de la petición actual, si hay alguna.

HTTP_ACCEPT_CHARSET
Los contenidos de la cabecera Accept-Charset: de la petición actual, si hay alguna. Por ejemplo: 'iso-8859-1,*,utf-8'.

HTTP_ENCODING
Los contenidos de la cabecera Accept-Encoding: de la petición actual, si la hay. Por ejemplo:
'gzip'.

HTTP_ACCEPT_LANGUAGE
Los contenidos de la cabecera Accept-Language: de la petición actual, si hay alguna. Por ejemplo: 'en'.

HTTP_CONNECTION
Los contenidos de la cabecera Connection: de la petición actual, si hay alguna. Por ejemplo: 'Keep-Alive'.

HTTP_HOST

Los contenidos de la cabecera Host: de la petición actual, si hay alguna.

HTTP_REFERER
La dirección de la página (si la hay) desde la que el navegador saltó a la página actual. Esto lo establece el navegador del usuario; no todos los navegadores lo hacen.

HTTP_USER_AGENT
Los contenidos de la cabecera User_Agent: de la petición actual, si hay alguna. Indica el navegador que se está utilizando para ver la página actual; p.ej. Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Entre otras cosas, se puede usar este valor con get_browser() para adaptar la funcionalidad de la página a las posibilidades del navegador del usuario.

REMOTE_ADDR
La dirección IP desde la que el usuario está viendo la página actual.

REMOTE_PORT
El puerto que se está utilizando en la máquina del usuario para comunicarse con el servidor web.

SCRIPT_FILENAME
La vía de acceso absoluta del script que se está ejecutando.

SERVER_ADMIN
El valor que se haya dado a la directiva SERVER_ADMIN (en Apache) en el fichero de configuración del servidor web. Si el script se está ejecutando en un servidor virtual, será el valor definido para dicho servidor virtual.

SERVER_PORT
El puerto del equipo servidor que está usando el servidor web para la comunicación. Para configuraciones por defecto, será ’80’; al usar SSL, por ejemplo, cambiará al puerto que se haya definido como seguro para HTTP.

SERVER_SIGNATURE
Una cadena que contiene la versión del servidor y el nombre del servidor virtual que es añadida a las páginas generadas por el servidor, si está característica está activa.

PATH_TRANSLATED
Vía de acceso basada en el sistema de ficheros- (no el directorio raíz del documento-) del script en cuestión, después de que el servidor haya hecho la conversión virtual-a-real.

SCRIPT_NAME
Contiene la vía de acceso del script actual. Es útil para páginas que necesitan apuntar a sí mismas.

REQUEST_URI

La URI que se dio para acceder a esta página; por ejemplo, '/index.html'.

Variables de PHP

Estas variables son creadas por el propio PHP.

argv
Array de argumentos pasados al script. Cuando el script se ejecuta desde la línea de comandos, esto da un acceso, al estilo de C, a los parámetros pasados en línea de comandos. Cuando se le llama mediante el método GET, contendrá la cadena de la petición.

argc
Contiene el número de parámetros de la línea de comandos pasados al script (si se ejecuta desde la línea de comandos).

PHP_SELF
El nombre del fichero que contiene el script que se esta ejecutando, relativo al directorio raíz de los documentos. Si PHP se está ejecutando como intérprete de línea de comandos, esta variable no está disponible.

HTTP_COOKIE_VARS
Un array asociativo de variables pasadas al script actual mediante cookies HTTP. Sólo está disponible si el seguimiento de variables ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

HTTP_GET_VARS
Un array asociativo de variables pasadas al script actual mediante el método HTTP GET. Sólo está disponible si --variable tracking-- ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

HTTP_POST_VARS
Un array asociativo de variables pasadas al script actual mediante el método HTTP POST. Sólo está disponible si --variable tracking-- ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos.

Ejemplo 4:

```
$a = 1;
include "b.inc";
```

Aquí, la variable $a dentro del script incluido b.inc. De todas formas, dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variables que se use dentro de una función está, por defecto, limitada al ámbito local de la función.

Ejemplo 5:

```
$a = 1; /* ámbito global */
Function Test () {
echo $a; /* referencia a una variable de ámbito local */
}
Test ();
```

Este script no producirá salida, ya que la orden echo utiliza una versión local de la variable $a, a la que no se ha asignado ningún valor en su ámbito. Puede que usted note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobreescritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función.

Ejemplo 6:

```
$a = 1;
$b = 2;
Function Sum () {
global $a, $b;
$b = $a + $b;
}
Sum ();
echo $b;
```

El script anterior producirá la salida "3". Al declarar $a y $b globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando el array $GLOBALS propio de PHP3.

El ejemplo anterior se puede reescribir así:

Ejemplo 7.

```
$a = 1;
$b = 2;
Function Sum () {
$GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum ();
```

```
echo $b;
```

El array $GLOBALS es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array.

Otra característica importante del ámbito de las variables es la variable static. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

Ejemplo 8.

```
Function Test () {
$a = 0;
echo $a;
$a++;
}
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a $a el valor 0 y representa un "0". La sentencia $a++, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina la variable $a desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable $a debe declararse como estática:

```
Function Test () {
static $a = 0;
echo $a;
$a++;
}
```

Ahora, cada vez que se llame a la función **Test(),** se representará el valor de $a y se incrementará.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática $count para saber cuándo parar:

Ejemplo 9.

```
Function Test () {
static $count = 0;
$count++;
echo $count;
if ($count < 10) {
Test ();
}
```

```
$count--;
}
```

Variables variables

A veces es conveniente tener nombres de variables variables. Dicho de otro modo, son nombres de variables que se pueden establecer y usar dinámicamente. Una variable normal se establece con una sentencia como:

Ejemplo 10:

```
$a = "hello";
```

Una variable variable toma el valor de una variable y lo trata como el nombre de una variable. En el ejemplo anterior, hello, se puede usar como el nombre de una variable utilizando dos signos de dólar.

Ejemplo 11:

```
$$a = "world";
```

En este momento se han definido y almacenado dos variables en el árbol de símbolos de PHP: $a, que contiene "hello", y $hello, que contiene "world". Es más, esta sentencia:

Ejemplo 12:

```
echo "$a ${$a}";
```

Produce el mismo resultado que:

Ejemplo 13.

```
echo "$a $hello";
```

p.ej. ambas producen el resultado: hello world.

Para usar variables variables con arrays, hay que resolver un problema de ambigüedad. Si se escribe $$a[1] el intérprete necesita saber si nos referimos a utilizar $a[1] como una variable, o si se pretendía utilizar $$a como variable y el índice [1] como índice de dicha variable. La sintaxis para resolver esta ambigüedad es: ${$a[1]} para el primer caso y ${$a}[1] para el segundo.

Variables externas a PHP

Formularios HTML (GET y POST)

Cuando se envía un formulario a un script PHP, las variables de dicho formulario pasan a estar automáticamente disponibles en el script gracias a PHP. Por ejemplo, consideremos el siguiente formulario:

Ejemplo 14.

```
<form action="foo.php3" method="post">
Name: <input type="text" name="name"><br>
<input type="submit">
</form>
```

Cuando es enviado, PHP creará la variable $name, que contendrá lo que sea que se introdujo en el campo Name: del formulario.

PHP también maneja arrays en el contexto de variables de formularios, pero sólo en una dimensión. Se puede, por ejemplo, agrupar juntas variables relacionadas, o usar esta característica para recuperar valores de un campo select input múltiple:

Ejemplo 15.

```
<form action="array.php" method="post">
Name: <input type="text" name="personal[name]"><br>
Email: <input type="text" name="personal[email]"><br>
Beer: <br>
<select multiple name="beer[]">
<option value="warthog">Warthog
<option value="guinness">Guinness
<option value="stuttgarter">Stuttgarter Schwabenbräu
</select>
<input type="submit">
</form>
```

Si la posibilidad de PHP de track_vars está activada, ya sea mediante la opción de configuración track_vars o mediante la directiva <?php_track_vars?>, las variables enviadas con los métodos POST o GET también se encontrarán en los arrays asociativos globales $HTTP_POST_VARS y $HTTP_GET_VARS.

IMAGE SUBMIT variable names

Cuando se envía un formulario, es posible usar una imagen en vez del botón submit estándar con una etiqueta como:

Ejemplo 16.

```
<input type=image src="image.gif" name="sub">
```

Cuando el usuario hace click en cualquier parte de la imagen, el formulario que la acompaña se transmitirá al servidor con dos variables adicionales, sub_x y sub_y. Estas contienen las coordenadas del click del usuario dentro de la imagen. Los más experimentados puede notar que los nombres de variable enviados por el navegador contienen un guión en vez de un subrayado (guión bajo), pero PHP convierte el guión en subrayado automáticamente.

6.4 Tipos de Datos.

PHP soporta los siguientes tipos:

• Entero

• Números en punto flotante

• Cadena

• Array

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución dependiendo del contexto en el que se utilice esa variable.

Si se quisiese obligar a que una variable se convierta a un tipo concreto, se podría forzar la variable o usar la función **settype()** para ello.

Nótese que una variable se puede comportar de formas diferentes en ciertas situaciones, dependiendo de qué tipo sea en ese momento. Para más información, vea la sección Conversión de Tipos.

6.4.1 Enteros

Los enteros se pueden especificar usando una de las siguientes sintaxis:

Ejemplo 1.

```
$a = 1234; # número decimal
$a = -123; # un número negativo
$a = 0123; # número octal (equivalente al 83 decimal)
$a = 0x12; # número hexadecimal (equivalente al 18 decimal)
```

6.4.2 Números en punto flotante

Los números en punto flotante ("double") se pueden especificar utilizando cualquiera de las siguientes sintaxis:

Ejemplo 1.

```
$a = 1.234; $a = 1.2e3;
```

6.4.3 Cadenas

Las cadenas de caracteres se pueden especificar usando uno de dos tipos de delimitadores.

Si la cadena está encerrada entre dobles comillas ("), las variables que estén dentro de la cadena serán expandidas (sujetas a ciertas limitaciones de interpretación). Como en C y en Perl, el carácter de barra invertida ("\") se puede usar para especificar caracteres especiales:

Tabla 6-1. Caracteres protegidos

secuencia	significado
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\\	Barra invertida
\$	Signo del dólar
\"	Comillas dobles
\[0-7]{1,3}	La secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal
\x[0-9A-Fa-f]{1,2}	La secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal

Se puede proteger cualquier otro carácter, pero se producirá una advertencia en el nivel de depuración más alto.

La segunda forma de delimitar una cadena de caracteres usa el carácter de comilla simple ("'"). Cuando una cadena va encerrada entre comillas simples, los únicos caracteres de escape que serán comprendidos son "\\" y "\'". Esto es por convenio, así que se pueden tener comillas simples y barras invertidas en una cadena entre comillas simples. Las variables no se expandirán dentro de una cadena entre comillas simples.

Otra forma de delimitar cadenas es usando la sintaxis de documento incrustado ("<<<"). Se debe proporcionar un identificador después de <<<, después la cadena, y después el mismo identificador para cerrar el entrecomillado.

6.4.4 Array.

Los arrays actualmente actúan tanto como tablas hash (arrays asociativos) como arrays indexados (Vectores).

Arrays unidimensionales.

PHP soporta tanto arrays escalares como asociativos. De hecho, no hay diferencias entre los dos. Se puede crear una array usando las funciones **list()** o **array()**, o se puede asignar el valor de cada elemento del array de manera explícita.

Ejemplo 1.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

También se puede crear un array simplemente añadiendo valores al array. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

Ejemplo 2.

```
$a[] = "hola"; // $a[2] == "hola"
$a[] = "mundo"; // $a[3] == "mundo"
```

Los arrays se pueden ordenar usando las funciones **asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort()** dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función **count().**
Se puede recorrer un array usando las funciones **next() y prev().** Otra forma habitual de recorrer un array es usando la función **each().**

Arrays Multidimensionales.

Los arrays multidimensionales son bastante simples actualmente. Para cada dimensión del array, se puede añadir otro valor [clave] al final:

Ejemplo 3.

```
$a[1] = $f; # ejemplos de una sola dimensión
$a["foo"] = $f;
$a[1][0] = $f; # bidimensional
$a["foo"][2] = $f; # (se pueden mezclar índices numéricos y
asociativos)
$a[3]["bar"] = $f; # (se pueden mezclar índices numéricos y
asociativos)
$a["foo"][4]["bar"][0] = $f; # tetradimensional!
```

En PHP3 no es posible referirse a arrays multidimensionales directamente dentro de cadenas. Por ejemplo, lo siguiente no tendrá el resultado deseado:

Ejemplo 4.

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: $a[3][bar]";
```

En PHP3, lo anterior tendrá la salida Esto no va a funcionar: Array[bar]. De todas formas, el operador de concatenación de cadenas se puede usar para solucionar esto:

Ejemplo 5.

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: " . $a[3][bar];
```

En PHP4, sin embargo, todo el problema se puede circunvenir encerrando la referencia al array (dentro de la cadena) entre llaves:

Ejemplo 6.

```
$a[3]['bar'] = 'Bob';
echo "Esto va a funcionar: {$a[3][bar]}";
```

Se pueden "rellenar" arrays multidimensionales de muchas formas, pero la más difícil de comprender es cómo usar el comando **array()** para arrays asociativos. Estos dos trozos de código rellenarán el array unidimensional de la misma manera:

Ejemplo 7.

```
# Ejemplo 1:
$a["color"] = "rojo";
$a["sabor"] = "dulce";
$a["forma"] = "redondeada";
$a["nombre"] = "manzana";
$a[3] = 4;
# Example 2:
$a = array(
"color" => "rojo",
"sabor" => "dulce",
"forma" => "redondeada",
"nombre" => "manzana",
3 => 4
);
```

La función **array()** se puede anidar para arrays multidimensionales:

Ejemplo 8.

```
<?
```

```
$a = array(
     "manzana" => array(
     "color" => "rojo",
     "sabor" => "dulce",
     "forma" => "redondeada"
     ),
     "naranja" => array(
     "color" => "naranja",
     "sabor" => "ácido",
     "forma" => "redondeada"
     ),
     "plátano" => array(
     "color" => "amarillo",
     "sabor" => "paste-y",
     "forma" => "aplatanada"
     )
);

echo $a["manzana"]["sabor"]; # devolverá "dulce"
?>
```

6.4.5 Objetos.

Inicialización de Objetos

Para inicializar un objeto, se usa la sentencia new para instanciar el objeto a una variable.

```
class foo {
function do_foo () {
echo "Doing foo.";
}
}
$bar = new foo;
$bar->do_foo();
```

Type juggling

PHP no requiere (o soporta) la declaración explícita del tipo en la declaración de variables; el tipo de una variable se determina por el contexto en el que se usa esa variable. Esto quiere decir que si se asigna un valor de cadena a la variable var, var se convierte en una cadena. Si después se asigna un valor entero a la variable var, se convierte en una variable entera.

Un ejemplo de conversión de tipo automática en PHP3 es el operador suma '+'. Si cualquiera de los operandos es un doble, entonces todos los operandos se evalúan como dobles, y el resultado será un doble. En caso contrario, los operandos se interpretarán como enteros, y el resultado será también un entero. Nótese que esto NO cambia los tipos de los operandos propiamente dichos; el único cambio está en cómo se evalúan los operandos.

```
$foo = "0"; // $foo es una cadena (ASCII 48)
$foo++; // $foo es la cadena "1" (ASCII 49)
$foo += 1; // $foo ahora es un entero (2)
$foo = $foo + 1.3; // $foo ahora es un doble (3.3)
$foo = 5 + "10 Cerditos Pequeñitos"; // $foo es entero (15)
$foo = 5 + "10 Cerditos"; // $foo es entero (15)
```

Si los últimos dos ejemplos anteriores parecen confusos, vea Conversión de cadenas.

Si se desea obligar a que una variable sea evaluada con un tipo concreto, mire la sección Forzado de tipos. Si se desea cambiar el tipo de una variable, vea la función settype().

Si quisiese probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

Nota: La posibilidad de una conversión automática a array no está definida actualmente.

```
$a = 1; // $a es un entero
$a[0] = "f"; // $a se convierte en un array, en el que $a[0]
vale "f"
```

Aunque el ejemplo anterior puede parecer que claramente debería resultar en que $a se convierta en un array, el primer elemento del cual es 'f', consideremos esto:

```
$a = "1"; // $a es una cadena
$a[0] = "f"; // ¿Qué pasa con los índices de las cadenas?
¿Qué ocurre?
```

Dado que PHP soporta indexación en las cadenas vía offsets usando la misma sintaxis que la indexación de arrays, el ejemplo anterior nos conduce a un problema: ¿debería convertirse $a en un array cuyo primer elemento sea "f", o debería convertirse "f" en el primer carácter de la cadena $a? Por esta razón, tanto en PHP 3.0.12 como en PHP 4.0b3-

RC4, el resultado de esta conversión automática se considera que no está definido. Los parches se están discutiendo, de todas formas.

Forzado de tipos

El forzado de tipos en PHP funciona como en C: el nombre del tipo deseado se escribe entre paréntesis antes de la variable a la que se pretende forzar.

```
$foo = 10; // $foo es un entero
$bar = (double) $foo; // $bar es un doble
```

Los forzados de tipo permitidos son:
• (int), (integer) - fuerza a entero (integer)
• (real), (double), (float) - fuerza a doble (double)
• (string) - fuerza a cadena (string)
• (array) - fuerza a array (array)
• (object) - fuerza a objeto (object)

Nótese que las tabulaciones y espacios se permiten dentro de los paréntesis, así que los siguientes ejemplos son funcionalmente equivalentes:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Puede no ser obvio que ocurrirá cuando se fuerce entre ciertos tipos. Por ejemplo, lo siguiente debería ser tenido en cuenta.

Cuando se fuerza el cambio de un escalar o una variable de cadena a un array, la variable se convertirá en el primer elemento del array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0]; // produce la salida 'ciao'
```

Cuando se fuerza el tipo de una variable escalar o de una cadena a un objeto, la variable se convertirá en un atributo del objeto; el nombre del atributo será 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar; // produce la salida 'ciao'
```

6.5 Estructuras de Control.

Todo archivo de comandos PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves.

Un grupo de sentencias es también una sentencia. En este capítulo se describen los diferentes tipos de sentencias.

if

La construcción if es una de las más importantes características de muchos lenguajes, incluido PHP.

Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura if que es similar a la de C:

```
if (expr)
     sentencia
```

Como se describe en la sección sobre expresiones, expr se evalúa a su valor condicional. Si exprese evalúa como TRUE, PHP ejecutará la sentencia, y si se evalúa como FALSE - la ignorará.

El siguiente ejemplo mostraría a es mayor que b si $a fuera mayor que $b:

```
if ($a > $b)
     print "a es mayor que b";
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula if. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este código mostraría a es mayor que b si $a fuera mayor que $b, y entonces asignaría el valor de $a a $b:

```
if ($a > $b) {
     print "a es mayor que b";
     $b = $a;
}
```

Las sentencias if se pueden anidar indefinidamente dentro de otras sentencias if, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

Else

A menudo queremos ejecutar una sentencia si se cumple una cierta condicion, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve else. else extiende una sentencia if para ejecutar una sentencia en caso de que la expresión en la sentencia if se evalúe como FALSE. Por ejemplo, el siguiente código mostraría a es mayor que b si $a fuera mayor que $b, y $a NO es mayor que b en cualquier otro caso:

```
if ($a > $b) {
     print "a es mayor que b";
} else {
     print "a NO es mayor que b";
}
```

La sentencia else se ejecuta solamente si la expresión if se evalúa como FALSE, y si hubiera alguna expresión elseif - sólo si se evaluaron también a FALSE (Ver elseif).

elseif

elseif, como su nombre sugiere, es una combinación de if y else. Como else, extiende una sentencia if para ejecutar una sentencia diferente en caso de que la expresión if original se evalúa como FALSE. No obstante, a diferencia de else, ejecutará esa expresión alternativa solamente si la expresión condicional elseif se evalúa como TRUE. Por ejemplo, el siguiente código mostraría a es mayorque b, a es igual a b o a es menor que b:

```
if ($a > $b) {
     print "a es mayor que b";
} elseif ($a == $b) {
     print "a es igual que b";
} else {
     print "a es mayor que b";
}
```

Puede haber varios elseifs dentro de la misma sentencia if. La primera expresión elseif (si hay alguna) que se evalúe como TRUE se ejecutaría. En PHP, también se puede escribir 'else if' (con dos palabras) y el comportamiento sería idéntico al de un 'elseif' (una sola palabra). El significado sintáctico es ligeramente distinto (si estas familiarizado con C, es el mismo comportamiento) pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia elseif se ejecuta sólo si la expresión if precedente y cualquier expresión elseif precedente se evalúan como FALSE, y la expresión elseif actual se evalúa como TRUE.

PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, if, while, for, y switch. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:) y cerrar-llave por endif;, endwhile;, endfor;, or endswitch;, respectivamente.

```
<?php if ($a==5): ?>
     A es igual a 5
<?php endif; ?>
```

En el ejemplo de arriba, el bloque HTML "A = 5" se anida dentro de una sentencia if escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si $a fuera igual a 5.

La sintaxis alternativa se aplica a else y también a elseif. La siguiente es una estructura if con elseif y else en el formato alternativo:

```
if ($a == 5):
     print "a es igual a 5";
     print "...";
elseif ($a == 6):
     print "a es igual a 6";
     print "!!!";
else:
     print "a no es ni 5 ni 6";
endif;
```

6.5.2 Bucles.

While

Los bucles while son los tipos de bucle más simples en PHP. Se comportan como su contrapartida en C.

La forma básica de una sentencia while es:

```
while (expr) sentencia
```

El significado de una sentencia while es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión while se evalúe como TRUE. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no

parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión while se evalúa como FALSE desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.

Como con la sentencia if, se pueden agrupar multiples sentencias dentro del mismo bucle while encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa:

```
while (expr): sentencia ... endwhile;
```

Los siguientes ejemplos son idénticos, y ambos imprimen números del 1 al 10:

```
$i = 1;
while ($i <= 10) {
print $i++; /* el valor impreso sería
$i antes del incremento
(post-incremento) */
}

$i = 1;
while ($i <= 10):
print $i;
$i++;
endwhile;
```

do..while

Los bucles do..while son muy similares a los bucles while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares while es que se garantiza la ejecución de la primera iteración de un bucle do..while (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle while regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como FALSE desde el principio la ejecución del bucle finalizará inmediatamente).

Hay una sola sintaxis para los bucles do..while:

```
$i = 0;
do {
print $i;
} while ($i>0);
for ($i = 1; $i <= 10; $i++) {
print $i;
}
```

```
/* ejemplo 2 */
for ($i = 1;;$i++) {
if ($i > 10) {
break;
}
print $i;
}
/* ejemplo 3 */
$i = 1;
for (;;) {
if ($i > 10) {
break;
}
print $i;
$i++;
}
```

```
for ($i = 1; $i <= 10; print $i, $i++) ;
```

Por supuesto, el primer ejemplo parece ser el mas elegante (o quizás el cuarto), pero uno puede descubrir que ser capaz de usar expresiones vacías en bucles for resulta útil en muchas ocasiones.

PHP también soporta la "sintaxis de dos puntos" alternativa para bucles for.

```
for (expr1; expr2; expr3): sentencia; ...; endfor;
```

Otros lenguajes poseen una sentencia foreach para traducir un array o una tabla hash. PHP3 no posee tal construcción; PHP4 sí (ver foreach). En PHP3, se puede combinar while con las funciones list() y each() para conseguir el mismo efecto. Mirar la documentación de estas funciones para ver un ejemplo.

foreach

PHP4 (PHP3 no) incluye una construcción foreach, tal como perl y algunos otros lenguajes. Esto simplemente da un modo fácil de iterar sobre arrays. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

```
foreach(expresion_array as $value) sentencia
foreach(expresion_array as $key => $value) sentencia
```

La primera forma recorre el array dado por expresion_array. En cada iteración, el valor del elemento actual se asigna a $value y el puntero interno del array se avanza en una unidad (así en el siguiente paso, se estará mirando el elemento siguiente).

La segunda manera hace lo mismo, salvo que la clave del elemento actual será asignada a la variable $key en cada iteración.

Nota: Cuando foreach comienza su primera ejecución, el puntero interno a la lista (array) se reinicia automáticamente al primer elemento del array. Esto significa que no se necesita llamar a reset() antes de un bucle foreach.

Puede haber observado que las siguientes son funcionalidades idénticas:

```
reset( $arr );
while( list( , $value ) = each( $arr ) ) {
echo "Valor: $value<br>\n";
}
foreach( $arr as $value ) {
echo "Valor: $value<br>\n";
}
```

Las siguientes también son funcionalidades idénticas:

```
reset( $arr );
while( list( $key, $value ) = each( $arr ) ) {
echo "Key: $key; Valor: $value<br>\n";
}
```

Algunos ejemplos más para demostrar su uso:

```
/* foreach ejemplo 1: sólo valor*/
$a = array(1, 2, 3, 17);
foreach($a as $v) {
print "Valor actual de \$a: $v.\n";
}
/* foreach ejemplo 2: valor (con clave impresa para ilustrar)
*/
$a = array(1, 2, 3, 17);
$i = 0; /* sólo para propósitos demostrativos */
foreach($a as $v) {
print "\$a[$i] => $k.\n";
}
/* foreach ejemplo 3: clave y valor */
$a = array(
"uno" => 1,
"dos" => 2,
"tres" => 3,
```

```
"diecisiete" => 17
);
foreach($a as $k => $v) {
print "\$a[$k] => $v.\n";
}
```

break

break escapa de la estructuras de control iterante (bucle) actuales for, while, o switch.

break accepta un parámetro opcional, el cual determina cuantas estructuras de control hay que escapar.

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
if ($val == 'stop') {
break; /* You could also write 'break 1;' here. */

}
echo "$val<br>\n";
}
/* Using the optional argument. */
$i = 0;
while (++$i) {
switch ($i) {
case 5:
echo "At 5<br>\n";
break 1; /* Exit only the switch. */
case 10:
echo "At 10; quitting<br>\n";
break 2; /* Exit the switch and the while. */
default:
break;
}
}
```

switch

La sentencia switch es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con nuchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia switch.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias if, y el otro usa la sentencia switch:

```
if ($i == 0) {
print "i es igual a 0";
}
if ($i == 1) {
print "i es igual a 1";
}
if ($i == 2) {
print "i es igual a 2";
}
switch ($i) {
case 0:
print "i es igual a 0";
break;
case 1:
print "i es igual a 1";
break;
case 2:
print "i es igual a 2";
break;
}
```

Es importante entender cómo se ejecuta la sentencia switch para evitar errores. La sentencia switch ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia case con un valor que coincide con el valor de la expresión switch PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o la primera vez que vea una sentencia break. Si no se escribe una sentencia break al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
switch ($i) {
case 0:
print "i es igual a 0";
case 1:
print "i es igual a 1";
case 2:
print "i es igual a 2";
}
```

Aquí, si $i es igual a 0, ¡PHP ejecutaría todas las sentecias print! Si $i es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $i es igual a 2, se obtendría la conducta 'esperada' y solamente se mostraría 'i es igual a 2'. Así, es importante no olvidar las sentencias break (incluso aunque pueda querer evitar escribirlas intencionadamente en ciertas circunstancias).

En una sentencia switch, la condición se evalúa sólo una vez y el resultado se compara a cada sentencia case. En una sentencia elseif, la condición se evalúa otra vez. Si tu condición es más complicada que una comparación simple y/o está en un bucle estrecho, un switch puede ser más rápido.

La lista de sentencias de un case puede también estar vacía, lo cual simplemente pasa el control a la lista de sentencias del siguiente case.

```
switch ($i) {
case 0:
case 1:
case 2:
print "i es menor que 3, pero no negativo";
break;
case 3:
print "i es 3";
}
```

Un case especial es el default case. Este case coincide con todo lo que no coincidan los otros case. Por ejemplo:

```
switch ($i) {
case 0:
print "i es igual a 0";
break;
case 1:
print "i es igual a 1";
break;
case 2:
print "i es igual a 2";
break;
default:
print "i no es igual a 0, 1 o 2";
}
```

6.5.3 Funciones.

require()

La sentencia require() se sustituye a sí misma con el archivo especificado, tal y como funciona la directiva #include de C.

Un punto importante sobre su funcionamiento es que cuando un archivo se incluye con include() o se requiere con require()), el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

require() no es en realidad una función de PHP; es más una construcción del lenguaje. Está sujeta a algunas reglas distintas de las de funciones. Por ejemplo, require() no esta sujeto a ninguna estructura de control contenedora. Por otro lado, no devuelve ningún valor; intentar leer un valor de retorno de una llamada a un require() resulta en un error del intérprete.

A diferencia de include(), require() siempre leerá el archivo referenciado, incluso si la línea en que está no se ejecuta nunca. Si se quiere incluir condicionalmente un archivo, se usa include(). La sentencia conditional no afecta a require(). No obstante, si la línea en la cual aparece el require() no se ejecuta, tampoco se ejecutará el código del archivo referenciado.

De forma similar, las estructuras de bucle no afectan la conducta de require(). Aunque el código contenido en el archivo referenciado está todavía sujeto al bucle, el propio require() sólo ocurre una vez.

Esto significa que no se puede poner una sentencia require() dentro de una estructura de bucle y esperar que incluya el contenido de un archivo distinto en cada iteración. Para hacer esto, usa una sentencia include().

En PHP3, es posible ejecutar una sentencia return dentro de un archivo referenciado con require(), en tanto en cuanto esa sentencia aparezca en el ámbito global del archivo requerido (require()). No puede aparecer dentro de ningún bloque (lo que siginifica dentro de llaves({})). En PHP4, no obstante, esta capacidad ha sido desestimada. Si se necesita esta funcionalidad.

include()

La sentencia include() incluye y evalúa el archivo especificado.

Si "URL fopen wrappers" esta activada en PHP (como está en la configuración inicial), se puede especificar el fichero que se va a incluir usando una URL en vez de un fichero local (con su Path).

Un punto importante sobre su funcionamiento es que cuando un archivo se incluye con include() o se requiere con require(), el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado

como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

Esto sucede cada vez que se encuentra la sentencia include(), así que se puede usar una sentencia include() dentro de una estructura de bucle para incluir un número de archivos diferentes.

```
$archivos = array ('primero.inc', 'segundo.inc',
'tercero.inc');
for ($i = 0; $i < count($archivos); $i++) {
include $archivos[$i];
}
```

include() difiere de require() en que la sentencia include se re-evalúa cada vez que se encuentra (y sólo cuando está siendo ejecutada), mientras que la sentencia require() se reemplaza por el archivo referenciado cuando se encuentra por primera vez, se vaya a evaluar el contenido del archivo o no (por ejemplo, si está dentro de una sentencia if cuya condición evaluada es falsa).
Debido a que include() es una construcción especial del lenguaje, se debe encerrar dentro de un bloque de sentencias si está dentro de un bloque condicional.

```
/* Esto es ERRÓNEO y no funcionará como se desea. */
if ($condicion)
include($archivo);
else
include($otro);
/* Esto es CORRECTO. */
if ($condicion) {
include($archivo);
} else {
include($otro);
}
```

En ambos, PHP3 y PHP4, es posible ejecutar una sentencia return dentro de un archivo incluido con include(), para terminar el procesado de ese archivo y volver al archivo de comandos que lo llamó.

Existen algunas diferencias en el modo en que esto funciona, no obstante. La primera es que en PHP3, return no puede aparecer dentro de un bloque a menos que sea un bloque de función, en el cual return se aplica a esa función y no al archivo completo. En PHP4, no obstante, esta restricción no existe. También, PHP4 permite devolver valores desde archivos incluidos con include(). Se puede capturar el valor de la llamada a include() como se haría con una función normal. Esto genera un error de intérprete en PHP3.

6.6 Manejo de Arreglos.

Arrays

Los arrays actualmente actúan tanto como tablas hash (arrays asociativos) como arrays indexados (vectores).

Arrays unidimensionales

PHP soporta tanto arrays escalares como asociativos. De hecho, no hay diferencias entre los dos. Se puede crear una array usando las funciones list() o array(), o se puede asignar el valor de cada elemento del array de manera explícita.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

También se puede crear un array simplemente añadiendo valores al array. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

```
$a[] = "hola"; // $a[2] == "hola"
$a[] = "mundo"; // $a[3] == "mundo"
```

Los arrays se pueden ordenar usando las funciones asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort() dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función count().

Se puede recorrer un array usando las funciones next() y prev(). Otra forma habitual de recorrer un array es usando la función each().

Arrays Multidimensionales

Los arrays multidimensionales son bastante simples actualmente. Para cada dimensión del array, se puede añadir otro valor [clave] al final:

```
$a[1] = $f; # ejemplos de una sola dimensión
$a["foo"] = $f;
$a[1][0] = $f; # bidimensional
$a["foo"][2] = $f; # (se pueden mezclar índices numéricos y
asociativos)
$a[3]["bar"] = $f; # (se pueden mezclar índices numéricos y
asociativos)
$a["foo"][4]["bar"][0] = $f; # tetradimensional!
```

En PHP3 no es posible referirse a arrays multidimensionales directamente dentro de cadenas. Por ejemplo, lo siguiente no tendrá el resultado deseado:

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: $a[3][bar]";
```

En PHP3, lo anterior tendrá la salida Esto no va a funcionar: Array[bar]. De todas formas, el operador de concatenación de cadenas se puede usar para solucionar esto:

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: " . $a[3][bar];
```

En PHP4, sin embargo, todo el problema se puede circunvenir encerrando la referencia al array (dentro de la cadena) entre llaves:

```
$a[3]['bar'] = 'Bob';
echo "Esto va a funcionar: {$a[3][bar]}";
```

Se pueden "rellenar" arrays multidimensionales de muchas formas, pero la más difícil de comprender es cómo usar el comando array() para arrays asociativos. Estos dos trozos de código rellenarán el array unidimensional de la misma manera:

```
# Ejemplo 1:
$a["color"] = "rojo";
$a["sabor"] = "dulce";
$a["forma"] = "redondeada";
$a["nombre"] = "manzana";
$a[3] = 4;

# Example 2:
$a = array(
"color" => "rojo",
"sabor" => "dulce",
"forma" => "redondeada",
"nombre" => "manzana",
3 => 4
);
```

La función array() se puede anidar para arrays multidimensionales:

```
<?
$a = array(
"manzana" => array(
"color" => "rojo",
"sabor" => "dulce",
```

```
"forma" => "redondeada"
),
"naranja" => array(
"color" => "naranja",
"sabor" => "ácido",
"forma" => "redondeada"
),
"plátano" => array(
"color" => "amarillo",
"sabor" => "paste-y",
"forma" => "aplatanada"
)
);
echo $a["manzana"]["sabor"]; # devolverá "dulce"
?>
```

6.7 Manejo de Funciones.

6.7.1 Funciones definidas por el usuario

Una función se define con la siguiente sintaxis:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
echo "Función de ejemplo.\n";
return $retval;
}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funciones y definiciones de clases.

En PHP3, las funciones deben definirse antes de que se referencien. En PHP4 no existe tal requerimiento.

PHP no soporta la sobrecarga de funciones, y tampoco es posible redefinir u ocultar funciones previamente declaradas.

PHP3 no soporta un número variable de parámetros, aunque sí soporta parámetros por defecto. PHP4 soporta ambos: ver Listas de longitud variable de parámetros y las referencias de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información.

6.7.2 Parámetros de las funciones

La información puede suministrarse a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.

PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto. Listas de longitud variable de parámetros sólo están soportadas en PHP4 y posteriores; ver Listas de longitud variable de parámetros y la referencia de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información. Un efecto similar puede conseguirse en PHP3 pasando un array de parámetros a la función:

```
function takes_array($input) {
echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

6.7.3 Pasar parámetros por referencia

Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si deseas permitir a una función modificar sus parámetros, debes pasarlos por referencia.

Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un ampersand (&) al nombre del parámetro en la definición de la función:

```
function add_some_extra(&$string) {
$string .= ' y algo más.';
}
$str = 'Esto es una cadena, ';
add_some_extra($str);
echo $str; // Saca 'Esto es una cadena, y algo más.'
```

Si deseas pasar una variable por referencia a una función que no toma el parámetro por referencia por defecto, puedes anteponer un ampersand al nombre del parámetro en la llamada a la función:

```
function foo ($bar) {
$bar .= ' y algo más.';
}
$str = 'Esto es una cadena, ';
foo ($str);
echo $str; // Saca 'Esto es una cadena, '
foo (&$str);
echo $str; // Saca 'Esto es una cadena, y algo más.'
```

6.7.4 Parámetros por defecto

Una función puede definir valores por defecto para los parámetros escalares estilo C++:

```
function makecoffee ($type = "cappucino") {
return "Hacer una taza de $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
La salida del fragmento anterior es:
Hacer una taza de cappucino.
Hacer una taza de espresso.
```

El valor por defecto tiene que ser una expresión constante, y no una variable o miembro de una clase.

En PHP 4.0 también es posible especificar unset como parámetro por defecto. Esto significa que el argumento no tomará ningún valor en absoluto si el valor no es suministrado.

Destacar que cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto; de otra manera las cosas no funcionará de la forma esperada. Considera el siguiente fragmento de código:

```
function makeyogurt ($type = "acidophilus", $flavour) {
return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora"); // No funcionará de la manera
esperada
```

La salida del ejemplo anterior es:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Haciendo un bol de mora.
```

Y ahora, compáralo con:

```
function makeyogurt ($flavour, $type = "acidophilus") {
return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora"); // funciona como se esperaba
```

La salida de este ejemplo es:

```
Haciendo un bol de acidophilus mora.
```

6.7.5 Lista de longitud variable de parámetros

PHP4 soporta las listas de longitud variable de parámetros en las funciones definidas por el usuario. Es realmente fácil, usando las funciones func_num_args(), func_get_arg(), y func_get_args().

No necesita de ninguna sintaxis especial, y las listasde parámetros pueden ser escritas en la llamada a la función y se comportarán de la manera esperada.

6.7.6 Devolver valores

Los valores se retornan usando la instrucción opcional return. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

```
function square ($num) {
return $num * $num;
}
echo square (4); // saca '16'
```

No puedes devolver múltiples valores desde una función, pero un efecto similar se puede conseguir devolviendo una lista.

```
function small_numbers() {
return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
```

6.8 Clases y Objetos.

6.8.1 Inicialización de Objetos

Para inicializar un objeto, se usa la sentencia new para instanciar el objeto a una variable.

```
class foo {
function do_foo () {
echo "Doing foo.";
}
}
$bar = new foo;
```

```
$bar->do_foo();
```

6.8.2 Type juggling

PHP no requiere (o soporta) la declaración explícita del tipo en la declaración de variables; el tipo de una variable se determina por el contexto en el que se usa esa variable. Esto quiere decir que si se asigna un valor de cadena a la variable var, var se convierte en una cadena. Si después se asigna un valor entero a la variable var, se convierte en una variable entera.

Un ejemplo de conversión de tipo automática en PHP3 es el operador suma '+'. Si cualquiera de los operandos es un doble, entonces todos los operandos se evalúan como dobles, y el resultado será un doble. En caso contrario, los operandos se interpretarán como enteros, y el resultado será también un entero. Nótese que esto NO cambia los tipos de los operandos propiamente dichos; el único cambio está en cómo se evalúan los operandos.

```
$foo = "0"; // $foo es una cadena (ASCII 48)
$foo++; // $foo es la cadena "1" (ASCII 49)
$foo += 1; // $foo ahora es un entero (2)
$foo = $foo + 1.3; // $foo ahora es un doble (3.3)
$foo = 5 + "10 Cerditos Pequeñitos"; // $foo es entero (15)
$foo = 5 + "10 Cerditos"; // $foo es entero (15)
```

Si los últimos dos ejemplos anteriores parecen confusos, vea Conversión de cadenas.

Si se desea obligar a que una variable sea evaluada con un tipo concreto, mire la sección Forzado de tipos. Si se desea cambiar el tipo de una variable, vea la función settype().
Si quisiese probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

Nota: La posibilidad de una conversión automática a array no está definida actualmente.

```
$a = 1; // $a es un entero
$a[0] = "f"; // $a se convierte en un array, en el que $a[0]
vale "f"
```

Aunque el ejemplo anterior puede parecer que claramente debería resultar en que $a se convierta en un array, el primer elemento del cual es 'f', consideremos esto:

```
$a = "1"; // $a es una cadena
```

```
$a[0] = "f"; // ¿Qué pasa con los índices de las cadenas?
¿Qué ocurre?
```

Dado que PHP soporta indexación en las cadenas vía offsets usando la misma sintaxis que la indexación de arrays, el ejemplo anterior nos conduce a un problema: ¿debería convertirse $a en un array cuyo primer elemento sea "f", o debería convertirse "f" en el primer carácter de la cadena $a? Por esta razón, tanto en PHP 3.0.12 como en PHP 4.0b3-RC4, el resultado de esta conversión automática se considera que no está definido. Los parches se están discutiendo, de todas formas.

6.8.3 Forzado de tipos

El forzado de tipos en PHP funciona como en C: el nombre del tipo deseado se escribe entre paréntesis antes de la variable a la que se pretende forzar.

```
$foo = 10; // $foo es un entero
$bar = (double) $foo; // $bar es un doble
```

Los forzados de tipo permitidos son:
• (int), (integer) - fuerza a entero (integer)
• (real), (double), (float) - fuerza a doble (double)
• (string) - fuerza a cadena (string)
• (array) - fuerza a array (array)
• (object) - fuerza a objeto (object)

Nótese que las tabulaciones y espacios se permiten dentro de los paréntesis, así que los siguientes ejemplos son funcionalmente equivalentes:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Puede no ser obvio que ocurrirá cuando se fuerce entre ciertos tipos. Por ejemplo, lo siguiente debería ser tenido en cuenta.

Cuando se fuerza el cambio de un escalar o una variable de cadena a un array, la variable se convertirá en el primer elemento del array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0]; // produce la salida 'ciao'
```

Cuando se fuerza el tipo de una variable escalar o de una cadena a un objeto, la variable se convertirá en un atributo del objeto; el nombre del atributo será 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar; // produce la salida 'ciao'
```

6.9 Manejo de Fecha y Hora.

El formato de la fecha de salida tipo string . Vea las opciones de formato más abajo. También hay varias constantes de fecha predefinidas que pueden usarse en su lugar, así por ejemplo DATE_RSS contiene la cadena de formato 'D, d M Y H:i:s'.

Los siguientes caracteres están reconocidos en el parámetro de cadena format		
Carácter format	Descripción	Ejemplo de valores devueltos
Día	---	---
d	Día del mes, 2 dígitos con ceros iniciales	01 a 31
D	Una representación textual de un día, tres letras	Mon hasta Sun
j	Día del mes sin ceros iniciales	1 a 31
l ('L' minúscula)	Una representación textual completa del día de la semana	Sunday hasta Saturday
N	Representación numérica ISO-8601 del día de la semana (añadido en PHP 5.1.0)	1 (para lunes) hasta 7 (para domingo)
S	Sufijo ordinal inglés para el día del mes, 2 caracteres	st, nd, rd o th. Funciona bien con j
w	Representación numérica del día de la semana	0 (para domingo) hasta 6 (para sábado)
z	El día del año (comenzando por 0)	0 hasta 365
Semana	---	---
W	Número de la semana del año ISO-8601, las semanas comienzan en lunes (añadido en PHP 4.1.0)	Ejemplo: 42 (la 42ª semana del año)
Mes	---	---
F	Una representación textual completa de un mes, como January o March	January hasta December
m	Representación numérica de una mes, con ceros iniciales	01 hasta 12
M	Una representación textual corta de un mes, tres letras	Jan hasta Dec
n	Representación numérica de un mes, sin ceros iniciales	1 hasta 12

t	Número de días del mes dado	28 hasta 31
Año	---	---
L	Si es un año bisiesto	1 si es bisiesto, 0 si no.
o	Número de año ISO-8601. Esto tiene el mismo valor que Y, excepto que si el número de la semana ISO (W) pertenece al año anterior o siguiente, se usa ese año en su lugar. (añadido en PHP 5.1.0)	Ejemplos: 1999 o 2003
Y	Una representación numérica completa de un año, 4 dígitos	Ejemplos: 1999 o 2003
y	Una representación de dos dígitos de un año	Ejemplos: 99 o 03
Hora	---	---
a	Ante meridiem y Post meridiem en minúsculas	am o pm
A	Ante meridiem y Post meridiem en mayúsculas	AM o PM
B	Hora Internet	000 hasta 999
g	Formato de 12 horas de una hora sin ceros iniciales	1 hasta 12
G	Formato de 24 horas de una hora sin ceros iniciales	0 hasta 23
h	Formato de 12 horas de una hora con ceros iniciales	01 hasta 12
H	Formato de 24 horas de una hora con ceros iniciales	00 hasta 23
i	Minutos, con ceros iniciales	00 hasta 59
s	Segundos, con ceros iniciales	00 hasta 59
u	Microsegundos (añadido en PHP 5.2.2)	Ejemplo: 654321
Zona Horaria	---	---
e	Identificador de zona horaria (añadido en PHP 5.1.0)	Ejemplos: UTC, GMT, Atlantic/Azores
I (i mayúscula)	Si la fecha está en horario de verano o no	1 si está en horario de verano, 0 si no.
O	Diferencia de la hora de Greenwich (GMT) en horas	Ejemplo: +0200
P	Diferencia con la hora de Greenwich (GMT) con dos puntos entre horas y minutos (añadido en PHP 5.1.3)	Ejemplo: +02:00
T	Abreviatura de la zona horaria	Ejemplos: EST, MDT ...
Z	Índice de la zona horaria en segundos. El índice	-43200 hasta 50400

	para zonas horarias al oeste de UTC siempre es negativo, y para aquellas al este de UTC es siempre positivo.	
Fecha/ Hora Completa	---	---
c	Fecha ISO 8601 (añadido en PHP 5)	2004-02-12T15:19:21+00:00
r	Fecha con formato » RFC 2822	Ejemplo: Thu, 21 Dec 2000 16:01:07 +0200
U	Segundos desde la Época Unix (1 de Enero del 1970 00:00:00 GMT)	Vea también time()

Los caracteres no reconocidos en la cadena de formato serán impresos tal cual. El formato Z siempre devolverá 0 cuando se usa gmdate().

Note: Ya que esta función sólo acepta marcas de tiempo de tipo integer el carácter de formato u sólo es útil cuando se usa la función date_format() con marcas de tiempo basadas en usuario creadas con date_create().

6.10 Manejo de Cadena de Caracteres.

Ejemplo #1 de date()

```
<?php
// Establecer la zona horaria predeterminada a usar. Disponib
le desde PHP 5.1
date_default_timezone_set('UTC');

// Imprime algo como: Monday
echo date("l");

// Imprime algo como: Monday 8th of August 2005 03:12:46 PM
echo date('l jS \of F Y h:i:s A');

// Imprime: July 1, 2000 is on a Saturday
echo "July 1, 2000 is on a " . date("l", mktime(0, 0, 0, 7, 1
, 2000));

/* Usar las constantes en el parámetro de formato */
// Imprime algo como: Mon, 15 Aug 2005 15:12:46 UTC
```

```
echo date(DATE_RFC822);

// Imprime algo como: 2000-07-01T00:00:00+00:00
echo date(DATE_ATOM, mktime(0, 0, 0, 7, 1, 2000));
?>
```

Puede prevenir que un carácter reconocido en la cadena de formato sea expandido escapándolo con una barra invertida precedente. Si el carácter con una barra invertida es ya una secuencia especial, necesitará escapar también la barra invertida.

Ejemplo #2 Escapar caracteres en date()

```
<?php
// imprime algo como: Wednesday the 15th
echo date("l \\t\h\e jS");
?>
```

Es posible usar date() y mktime() juntos para buscar fechas en el futuro o en el pasado.

Ejemplo #3 Ejemplo de date() y mktime()

```
<?php
$mañana        = mktime(0, 0, 0, date("m")  , date("d")+1, da
te("Y"));
$mes_anterior  = mktime(0, 0, 0, date("m")-1, date("d"),   da
te("Y"));
$año_siguiente = mktime(0, 0, 0, date("m"),   date("d"),   da
te("Y")+1);
?>
```

Note: Esto puede ser más fiable que añadir o sustraer simplemente el número de segundos de un día o mes a una marca de tiempo debido al horario de verano.

7.-PHP y MySQL.

7.1 Conexión al Servidor Local.

Una vez que tenemos creada la base de datos en nuestro servidor, el siguiente paso es conectarnos a la misma desde una página PHP. Para ello PHP nos proporciona una serie de instrucciones para acceder a bases de datos MySQL.

```
<!-- Manual de PHP de WebEstilo.com -->
<html>
<head>
   <title>Ejemplo de PHP</title>
</head>
<body>
<?php
function Conectarse()
{
   if (!
($link=mysql_connect("localhost","usuario","Password")))
   {
      echo "Error conectando a la base de datos.";
      exit();
   }
   if (!mysql_select_db("base_datos",$link))
   {
      echo "Error seleccionando la base de datos.";
      exit();
   }
   return $link;
}

$link=Conectarse();
echo "Conexión con la base de datos conseguida.<br>";

mysql_close($link); //cierra la conexion
?>
</body>
</html>
```

Al ejecutar la instrucción **mysql_connect** creamos un vínculo entre la base de datos y la pagina PHP, este vínculo será usado posteriormente en las consultas que hagamos a la base de datos.

Finalmente, una vez que hemos terminado de usar el vínculo con la base de datos, lo liberaremos con la instrucción **mysql_close** para que la conexión no quede ocupada.

7.2 Creación de la Base de Datos desde PHP.

7.2.1 mysql_create_db

Crea una base MySQL int mysql_create_db (string base_de_datos [, int identificador_de_enlace]) \linebreak mysql_create_db() intenta crear una base nueva en el servidor asociado al identificador de enlace.

Ejemplo 1. Ejemplo de MySQL create

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim") {
or die ("Could not connect");
}
if (mysql_create_db ("my_db")) {
print ("Database created successfully\n");
} else {
printf ("Error creating database: %s\n", mysql_error ());
}
?>
```

Por razones de compatibilidad puede usarse mysql_createdb() igualmente.

7.2.2 mysql_drop_db

Borra una base de datos MySQL int mysql_drop_db (string base_de_datos [, int identificador_de_enlace]) \linebreak Devuelve: verdadero si éxito, falso si error.

mysql_drop_db() intenta suprimir una base de datos completa del servidor asociado al identificador de enlace.

7.3 Conexión a la Base de Datos.

7.3.1 mysql_connect

Abre una conexión a un servidor MySQL int mysql_connect ([string server [, string usuario [, string password]]]) \linebreak Devuelve: Un identificador de enlace positivo si tiene exito, o falso si error.

mysql_connect()establece una conexión a un servidor MySQL. Todos los argumentos son opcionales, y si no hay , se asumen los valores por defecto (’localhost’, usuario propietario del proceso del servidor, password vacia).

El hostname puede incluir tambien un número de puerto . ej. "hostname:puerto" o un camino al socket ej. ":/camino/al/socket" para localhost.

Nota: Soporte para ":puerto" fue añadido en PHP 3.0B4. Soporte para ":/camino/al/socket" fue añadido en PHP 3.0.10.

En el caso de que se haga una llamada a mysql_connect() con los mismos argumentos, no se establecerá un nuevo enlace, sino que se devolverá el enlace ya abierto.

El enlace al servidor será cerrado tan pronto como la ejecución del script finalice, a menos que se cierre antes explicitamente llamando a mysql_close().

Ejemplo 1. Ejemplo de MySQL connect

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret") {
or die ("Could not connect");
}
print ("Connected successfully");
mysql_close ($link);
?>
```

7.4 Creación de Tablas desde PHP.

Ejemplo 1. Ejemplo de MySQL connect

```
$conexion=mysql_connect(host,usuario,contrasñea);
mysql_select_db(basededatos,$conexion);

mysql_query(tu consulta sql); no es necesario terminar en ;
las sentencias pero si tienen q ser de a una

mysql_query("CREATE TABLE `datos` (
`id` int(11) NOT NULL auto_increment,
`name` varchar(20) NOT NULL default '',
`value` text NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=33") ;

mysql_query(" INSERT INTO `datos` VALUES (1, 'nombreNegocio',
'Nombre')");
mysql_query(" INSERT INTO `datos` VALUES (2, 'telefono1', '00
```

```
000 0000')");
mysql_query(" INSERT INTO `datos` VALUES (3, 'telf1', '91 111
1111')");
mysql_query(" INSERT INTO `datos` VALUES (4, 'telf2', '92 222
2222')");

y asi sucecivamente con cualquier tipo de consutla

poddes agregar un or die despues e cada sentencia para
mostrar si ocurrio algun error

mysql_query(" INSERT INTO `datos` VALUES (4, 'telf2', '92 222
2222')")or die("no se pudo guardar el registro");
```

7.5 Funciones de MySQL .

7.5.1 mysql_affected_rows

Devuelve el número de filas afectadas de la última operación MySQL int mysql_affected_rows ([int identificador_de_enlace]) \linebreak mysql_affected_rows() devuelve el número de filas afectadas en la ultima sentencia INSERT, UPDATE o DELETE sobre el servidor asociado con el identificador de enlace especificado. Si el identificador de enlace no ha sido especificado, se asume por defecto el último enlace.
Si la última sentencia fue un DELETE sin clausula WHERE, todos los registros han sido borrados de la tabla pero esta función devolvera cero.

Este comando no es efectivo para las sentencias SELECT, sino sólo para las sentencias que modifican registros. Para conseguir el número de lineas devueltos por un SELECT, usar mysql_num_rows().

7.5.2 mysql_change_user

Cambia el usuario conectado en la conexión activa int mysql_change_user (string usuario, string password [, string base_de_datos [, int identificador_de_enlace]]) \linebreak mysql_change_user() cambia el usuario conectado en la actual conexión activa, o si se especifica, en la conexión determinada por el identificador de enlace. Si se especifica la base de datos, esta será la base por defecto después del cambio de usuario. Si la nueva combinación de usuario/ password no esta autorizada, el usuario actualmente conectado permanece activo.

7.5.3 mysql_close.

Cierra el enlace con MySQL int mysql_close ([int identificador_de_enlace]) \linebreak Devuelve: verdadero si exito, falso si error.

mysql_close() cierra el enlace con la base MySQL que esta asociada con el identificador de enlace especificado. Si no se especifica el identificador de enlace, se asume por defecto el último enlace.

mysql_close() no cerrará los enlaces persistentes generados con mysql_pconnect().

Ejemplo 1. Ejemplo de MySQL close

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret") {
or die ("Could not connect");
}
print ("Connected successfully");
mysql_close ($link);
?>
```

7.5.4 mysql_connect.

Abre una conexión a un servidor MySQL int mysql_connect ([string server [, string usuario [, string password]]]) \linebreak Devuelve: Un identificador de enlace positivo si tiene exito, o falso si error.

mysql_connect()establece una conexión a un servidor MySQL. Todos los argumentos son opcionales, y si no hay , se asumen los valores por defecto ('localhost', usuario propietario del proceso del servidor, password vacia).

El hostname puede incluir tambien un número de puerto . ej. "hostname:puerto" o un camino al socket ej. ":/camino/al/socket" para localhost.

En el caso de que se haga una llamada a mysql_connect() con los mismos argumentos, no se establecerá un nuevo enlace, sino que se devolverá el enlace ya abierto.

El enlace al servidor sera cerrado tan pronto como la ejecución del script finalice, a menos que se cierre antes explicitamente llamando a mysql_close().

Ejemplo 1. Ejemplo de MySQL connect

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret") {
```

```
or die ("Could not connect");
}
print ("Connected successfully");
mysql_close ($link);
?>
```

7.5.5 mysql_create_db

Crea una base MySQL int mysql_create_db (string base_de_datos [, int identificador_de_enlace]) \linebreak mysql_create_db() intenta crear una base nueva en el servidor asociado al identificador de enlace.

Ejemplo 1. Ejemplo de MySQL create

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim") {
or die ("Could not connect");
}
if (mysql_create_db ("my_db")) {
print ("Database created successfully\n");
} else {
printf ("Error creating database: %s\n", mysql_error ());
}
?>
```

Por razones de compatibilidad puede usarse mysql_createdb() igualmente.

7.5.6 mysql_data_seek

Mueve el puntero interno int mysql_data_seek (int id_resultado, int numero_de_fila) \ linebreak Devuelve: verdadero si exito, falso si error.

mysql_data_seek() mueve el puntero de fila interno a la fila especificada para el identificador de resultado. La próxima llamada a mysql_fetch_row() devolverá esa fila. numero_de_fila empieza en 0.

Ejemplo 1. Ejemplo de MySQL data seek

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim") {
or die ("Could not connect");
}
mysql_select_db ("samp_db") {
or die ("Could not select database");
}
```

```
$query = "SELECT last_name, first_name FROM friends";
$result = mysql_query ($query) {
or die ("Query failed");
}
# fetch rows in reverse order
for ($i = mysql_num_rows ($result) - 1; $i >=0; $i--) {
if (!mysql_data_seek ($result, $i)) {
printf ("Cannot seek to row %d\n", $i);
continue;
}
if(!($row = mysql_fetch_object ($result)))
continue;
printf ("%s %s<BR>\n", $row->last_name, $row->first_name);
}
mysql_free_result ($result);
?>
```

7.5.7 mysql_db_query

Envia una sentencia MySQL al servidor int mysql_db_query (string base_de_datos, string sentencia [, int identificador_de_enlace]) \linebreak Devuelve: Un identificador de resultado positivo o falso si error.

mysql_db_query() selecciona una base y ejecuta una sentencia en ella. Si el identificador de enlace no ha sido especificado, la función intenta encontrar un enlace abierto al servidor MySQL y si no lo encuentra, intetará crear uno como si fuera llamado mysql_connect() sin argumentos

7.5.8 mysql_drop_db

Borra una base de datos MySQL int mysql_drop_db (string base_de_datos [, int identificador_de_enlace]) \linebreak Devuelve: verdadero si éxito, falso si error.

mysql_drop_db() intenta suprimir una base de datos completa del servidor asociado al identificador de enlace.

7.5.9 mysql_errno

Deuelve el número del mensaje de error de la última operación MySQL int mysql_errno ([int identificador_de_enlace]) \linebreak Los errores devueltos por mySQL no indican los warnings. Usar estas funciones para encontrar el número de error.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

7.5.10 mysql_error

Devuelve el texto del mensaje de error de la última operación MySQL string mysql_error ([int identificador_de_enlace]) \linebreak Los errores devueltos por mySQL no indican los warnings. Usar estas funciones para encontrar el número de error.


```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```


7.5.11 mysql_fetch_array

Extrae la fila de resultado como una matriz asociativa array mysql_fetch_array (int id_resultado [, int tipo_de_resultado]) \linebreak Devuelve una matriz que corresponde a la sentencia extraida, o falso si no quedan más filas.

mysql_fetch_array() es una versión extendida de mysql_fetch_row(). Además de guardar los datos en el indice numerico de la matriz, guarda también los datos en los indices asociativos, usando el nombre de campo como clave.

Si dos o más columnas del resultado tienen el mismo nombre de campo, la última columna toma la prioridad. Para acceder a la(s) otra(s) columna(s) con el mismo nombre, se debe escificar el índice numerico o definir un alias para la columna.

La función mysql_fetch_array() no es significativemente mas lenta que mysql_fetch_row(), sin embargo tiene un valor añadido importante.

El segundo argumento opcional tipo_de_resultado en mysql_fetch_array() es una constante y puede tomar los valores siguientes: MYSQL_ASSOC, MYSQL_NUM, y YSQL_BOTH. (Esta funcionalidad fue añadida en PHP 3.0.7) Para más detalles, ver también mysql_fetch_row().

Ejemplo 1. mysql fetch array

```
<?php
mysql_connect($host,$user,$password);
$result = mysql_db_query("database","select * from table");
while($row = mysql_fetch_array($result)) {
echo $row["user_id"];
echo $row["fullname"];
}
mysql_free_result($result);
?>
```

7.5.12 mysql_fetch_field

Extrae la información de una columna y la devuelve como un objeto. object mysql_fetch_field (int id_resultado [, int salto]) \linebreak Devuelve un objeto que contiene la información del campo.

Puede usarse mysql_fetch_field() para obtener información sobre campos en un resultado. Si no se especifica el salto, se extrae el siguiente campo que todavia no ha sido extraido. con mysql_fetch_field().

Las propiedades del objeto son:
• name - nombre de la columna
• table - name de la tabla a la que pertenece la columna
• max_length - longitud máxima de la columna
• not_null - 1 si la columna no puede contener un valor nulo
• primary_key - 1 si la columna es clave primaria
• unique_key - 1 si la columna es clave unica
• multiple_key - 1 si la columna es clave no unica
• numeric - 1 si la columna es numerica
• blob - 1 si la columna es un BLOB
• type - el tipo de la columna
• unsigned - 1 si la columna es unsigned
• zerofill - 1 si la columna es zero-filled

8 Fuentes Bibliografia y Libros.

8.1 Libros:

López Quijado, José. Domine php y MySQL, 3ª Edición, México DF, AlfaOmega, 2007.

Guitierrez Abraham – Bravo, Ginés. PHP5 a través de ejemplos, 3ª Edición, México DF, AlfaOmega, 2005.

Pavón Puertas Jacobo, Creación de un portal con PHP y MySQL, 2ª edición, México DF, AlfaOmega, 2007.

8.2 En Internet:

PHP fundation : http://php.net Febrero 2010

MySQL: http://mysql.com Febrero 2010

Apache Friends: http://www.apachefriends.org/es/xampp.html Febrero 2010

Brainbell: http://www.brainbell.com/tutors/php/php_mysql Febrero 2010